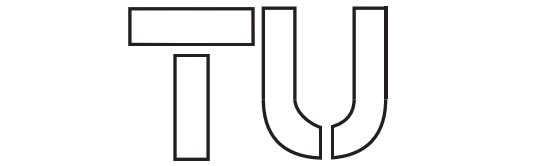


TECHNISCHE UNIVERSITÄT WIEN

Dissertation

Practical CG Resarch, Game Development and the European Union GameTools Project

ausgeführt
zum Zwecke der Erlangung des akademischen Grades
eines Doktors der technischen Wissenschaften

unter der Leitung von
Univ.Prof. Dipl.-Ing. Dr.techn. Werner Purgathofer,
Institut 186 für Computergraphik und Algorithmen,
und unter Mitwirkung von
Univ.-Ass. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer

eingereicht
an der Technischen Universität Wien,
Fakultät für Technische Naturwissenschaften und Informatik,

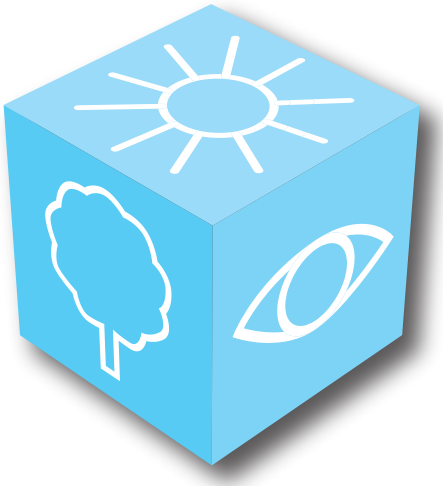
von
Dipl.-Ing. Markus Giegl,
Matrikelnummer 8725379,
Dampfschiffstraße 14/11,
A-1030 Wien, Österreich,
geboren am 21. September 1968 in Wien.

Wien, im Mai 2007.

Markus Giegl (PhD Thesis)

Practical CG Resarch, Game Development and the European Union GameTools Project

GEOMETRY – VISIBILITY – ILLUMINATION



GAMETOOLS

www.cg.tuwien.ac.at/research/vr/mg, www.gametools.org
giegl@cg.tuwien.ac.at, giegl@arscreat.com

Reviewer
Werner Purgathofer
László Szirmay-Kalos
Michael Wimmer

Abstract

Computer graphics today plays an important role in 3D computer and video game creation, since all the images which are presented to the player are based on principles and algorithms devised from computer graphics research.

One important aspect with regards to the believability and consecutively immersion of players within game worlds are shadows. Creating high–quality dynamic shadows in real-time is still a very open field of research. One real-time shadowing approach that is very appealing due to its apparent elegant simplicity is shadow mapping. The term “apparent” is used deliberately here, since the elegance of shadow mapping comes at a price, namely the discretization of the first-hit visibility problem to a grid storing occluder depth information with regards to the light source. In practice this often leads to shadowing artifacts, due to too little information being available to answer the visibility query with enough accuracy. In this thesis, a new family of shadow mapping algorithms is presented, which address the problem of the shadow map not containing enough information, This is done without requiring a shadow map grid too large to be stored in memory and which cannot be filled with depth information in real time. Based on a basic brute-force approach, two smart algorithms are presented, which work in a manner adaptive to the resolution requirements of the scene, speeding up the process by at least an order of magnitude.

Another problem with relevance to the practical application of computer graphics to games is how to render a large number of objects fast enough to give the player a high level of responsiveness (and thereby, again, immersion). Noticing that due to perspective shortening objects farther away from the player do not need as much detail as objects which are nearer, computer graphics has come up with the concept of level of detail (LOD), where farther away objects use simpler representations; if done correctly, this leads to a large increase in rendering speed. While LOD has been studied extensively in computer graphics, one problem of the LOD technique dominating practical application, discrete LOD, has been largely ignored: In discrete LOD the simpler representations of an object are, as the name implies, discrete, i.e. they exist independently of each other. Switching between representations of different complexity by simply switching the representation in use leads to a disruptive discontinuity in player perception. This thesis presents a practical algorithm, that addresses this problem.

Linked to this research in the computer graphics fields of shadows and level of detail, is the case study of the GameTools Project, a Europe-wide project, funded by the European Union, which had as its agenda to bring the results

of current computer graphics research done within the project to European game developing companies and companies from neighboring fields. The author holds the position of Community Manager within the GameTools Project, and presents an overview over the project and a more detailed view of his work as Community Manager.

Kurzfassung

Acknowledgements

Thanks go out to the following people:

My wife, Gabriele Giegl (aka Gab - as in “mind the...”), for putting up with the stress of me juggling the two jobs of GameTools Community Manager and computer graphics researcher.

Werner Purgathofer (aka A-Man) for agreeing to let me do my two jobs from home without any problems.

Michael Wimmer (aka Reverso), for his knowledge of and people in the CG field.

László Szirmay-Kalos, for agreeing to review this thesis.

Oliver Mattausch (aka CanDo Man), for many interesting and entertaining discussions, about life, the universe and everything.

Attila Neumann (aka Mr Brain), for many entertaining questions during institute Konversatorium.

Alexander Wilkie (aka TG) - for agreeing to differ.

The institute secretaries, Anita Mayerhofer (aka Anitator) and Andrea Fübi, for always being friendly and reliable.

The institute technicians (in alphabetical order), Alexander Piskernik, Stephan Plepelits and Andreas Weiner, for keeping the system functioning within specified parameters.

And finally my son, Raffel Adrian Salvador Ismael Gabriel Vincent-Emmanuel Immanuel Manuel Giegl (aka Bubu) - for just being there :-)

This research was supported by the EU in the scope of the
European Union GameTools Project
(www.gametools.org)
(IST-2-004363)

Contents

Abstract i

Kurzfassung iii

Acknowledgements v

1 Introduction 1
- 1.1 In the beginning . . . 1
- 1.2 Publications about this work . . . 1

2 Shadowing 3
- 2.1 Introduction . . . 3
- 2.2 Abbreviations . . . 3
- 2.3 Shadowing Techniques . . . 3
- 2.4 Shadowing in this Thesis . . . 6
- 2.5 Shadow Algorithms . . . 6
  - 2.5.1 Planar Shadows . . . 6
  - 2.5.2 Projected Shadows . . . 8
  - 2.5.3 Blob Shadows . . . 10
  - 2.5.4 Lightmaps . . . 10
  - 2.5.5 Shadow Volumes . . . 11
  - 2.5.6 Shadow Maps . . . 13
  - 2.5.7 PRT and Spherical Harmonics Lighting . . . 14
  - 2.5.8 Beyond Spherical Harmonics . . . 15
  - 2.5.9 Further Reading . . . 17
- 2.6 A Closer Look at Shadow Mapping . . . 17
  - 2.6.1 The Shadow Map Aliasing Problem . . . 18
  - 2.6.2 The Shadow Map Biasing Problem . . . 20
- 2.7 Previous Work about Shadow Mapping . . . 20

3 Virtual Shadow Maps 23

3.1 Introduction . . . 23
3.2 Abbreviations . . . 24
3.3 Virtual Tiled Shadow Mapping . . . 24
3.3.1 Creating the Shadow Map Tiles . . . 25
3.3.2 Multi-Pass Shadowing . . . 25
3.3.3 Deferred Shadowing . . . 26
3.4 Shadow Result Texture . . . 29
3.5 Combination with Shadow Map Filtering . . . 30
3.6 Problems with Brute Force Refinement . . . 30
3.7 Conclusion . . . 30

4 Queried Virtual Shadow Maps **32**
4.1 Introduction . . . 32
4.2 Abbreviations . . . 33
4.3 Queried Virtual Shadow Mapping . . . 34
4.3.1 Smart Refinement Preferred . . . 34
4.3.2 Queried Refinement: GPU Friendly & Smart . . . 35
4.4 Jump Optimizations . . . 36
4.4.1 Maximum Refinement Jump . . . 36
4.4.2 Opportunity Jump . . . 37
4.5 Results . . . 38
4.5.1 Extensions and Optimizations . . . 42
4.6 Hardware Extensions for Better GPU to CPU Feedback . . . 44
4.7 Conclusion . . . 44
4.8 QVSM Core Algorithm . . . 48

5 Fitted Virtual Shadow Maps **51**
5.1 Introduction . . . 51
5.2 Abbreviations . . . 52
5.3 Fitted Virtual Shadow Mapping . . . 52
5.4 Eye-Space Depth Buffer . . . 54
5.5 Shadow Map Tile Mapping Map - Motivation . . . 54
5.6 Shadow Map Tile Mapping Map - Technical Details . . . 57
5.7 Shadow Map Tile Grid Creation . . . 58
5.8 Shadow Map Tile Grid Pyramid Creation . . . 60
5.9 Shadow Map Tile Grid Pyramid Traversal . . . 61
5.10 Apply Shadow to Scene . . . 62
5.11 Quality vs Performance Parameter . . . 62
5.12 Shadow Map Tile Texture Size Optimization . . . 63
5.13 Handling Semitransparent Objects . . . 64

5.14 Results . . . 65
5.15 Conclusion . . . 69
5.16 Influence of ξ . . . 70
5.17 SMTMM Creation Pixel Shader . . . 73

6 Discrete LODs **75**
6.1 Introduction . . . 75
6.2 LOD Switching . . . 76
6.2.1 Hard Switching . . . 76
6.2.2 Late Switching . . . 76
6.2.3 LOD Blending in Image Space . . . 77
6.2.4 Geomorphing . . . 78
6.3 Popping Examples in Current Games . . . 78

7 Unpopping **86**
7.1 Introduction . . . 86
7.2 A New Method for LOD blending . . . 87
7.3 Discussion . . . 88
7.3.1 Silhouette Mismatch . . . 88
7.3.2 Depth Discontinuity . . . 89
7.3.3 z-Fighting . . . 89
7.4 Practical Application . . . 89
7.4.1 Transition Time . . . 89
7.4.2 Animated LODs . . . 90
7.4.3 Combination with Geomorphing . . . 90
7.5 Blending Different Objects . . . 90
7.6 Electronic Material . . . 90
7.7 Conclusion . . . 91
7.8 Unpopping Screenshots . . . 91

8 The European Union GameTools Project **95**
8.1 Introduction . . . 95
8.2 Abbreviations . . . 96
8.3 The European Union GameTools Project . . . 96
8.3.1 3D API . . . 98
8.3.2 3D Engine(s) . . . 99
8.4 GTP Research Areas . . . 100
8.4.1 Visibility (WP3) . . . 100
8.4.2 Geometry and Plants (WP4) . . . 101
8.4.3 Global Illumination and Effects (WP5) . . . 101
8.5 GTP Special Interest Group . . . 102

8.5.1 SIG Membership Agreement . . . 102
8.5.2 GameTools Evaluation License ? . . . 104
8.5.3 GTP SIG Questionnaires . . . 105
8.5.4 SIG Members . . . 105
8.5.5 GTP Software Repository . . . 111
8.6 GTP Support Forum . . . 114
8.7 GTP Advertising . . . 114
8.7.1 GTP logo . . . 114
8.7.2 GTP webpage . . . 116
8.7.3 Game Developer Conference Europe 2005 . . . 117
8.7.4 GTP Demo Games . . . 118
8.7.5 Eurographics 2006 . . . 119
8.7.6 gamasutra.com et al . . . 119
8.7.7 Resfest 2006 . . . 120
8.7.8 GTP Game Developer Magazine Ad . . . 120
8.8 GTP Demo Games . . . 121
8.8.1 Jungle Rumble . . . 121
8.8.2 Have U Seen My Shadow . . . 121
8.8.3 Penta-G . . . 122
8.9 Why GTP Technology is Not Free . . . 124
8.9.1 Free as in No Cost ? . . . 124
8.9.2 Public Domain Libraries . . . 124
8.9.3 Commercial vs Open Source . . . 125
8.9.4 Graphics Libraries . . . 127
8.9.5 The GameTools Libraries . . . 127
8.9.6 Future Costs . . . 129
8.9.7 Conclusion: GTP Technology is Not Free . . . 129
8.10 Taking Stock: An Early Postmortem . . . 129

9 Game Development and the Game Industry 146

10 Why Buying a Computer Game is Not Like Going to the Movies 147
10.1 Recognizing Creative Minds . . . 147
10.2 Communication . . . 148
10.3 Pricing . . . 148
10.4 Social Activity . . . 149
10.5 Ease of Picking a Good One . . . 149
10.6 Fun at Picking a Bad One . . . 149
10.7 Time Investment . . . 149
10.8 Language Barrier . . . 150

10.9 Controls 150
10.10 Bugfixes 150
10.11 Bootlegging 150
10.12 How to Improve the Situation 151

11 The Quality Frontier in 3D Games **152**

12 Why Computer Games are Seldom Art **154**

Bibliography **157**

Curriculum vitae **163**

List of Figures

2.1 Shadow Mapping 17
2.2 Visualization of projection and perspective aliasing. 19

3.1 VSM: Virtual Tiled Shadow Mapping 27
3.2 VTSM: Multi-pass vs Deferred Shadow Mapping 28

4.1 QVSM: Comparison with Conventional Shadow Mapping 32
4.2 QVSM: Forest Test Scene 33
4.3 QVSM: Comparison with Conventional Shadow Mapping 35
4.4 QVSM: Projection Aliasing 37
4.5 QVSM: Quality & Performance Comparison 38
4.6 QVSM: Performance Comparison along Path with VTSM 39
4.7 QVSM: Threshold Parameter η_{min} 39
4.8 QVSM: Performance Gain from Jump Optimizations 40
4.9 QVSM: Quality Comparison in Forest Test Scene 46
4.10 QVSM: Projection Aliasing 47

5.1 FVSM: Comparison with Conventional Shadow Mapping 51
5.2 FVSM Quality in Winter Forest 53
5.3 FVSM: SMTMM Creation 55
5.4 FVSM: Performance Comparison with VSM 66
5.5 FVSM: Tile Creation Modes Comparison 66
5.6 FVSM: Screen Space Rect Optimization 67
5.7 FVSM: Influence of ξ 70

6.1 LOD: Popping in Far Cry 80
6.2 LOD: Popping in Far Cry 81
6.3 LOD: Popping in F.E.A.R. 82
6.4 LOD: Popping in F.E.A.R. 83
6.5 LOD: Popping in Half-Life 2 84
6.6 LOD: Popping in Half-Life 2 85

7.1 Unpopping: Transition between two LODs 87
7.2 Unpopping: Render State Settings during LOD Blend 88
7.3 Unpopping: Palm Tree - Popping 92
7.4 Unpopping: Palm Tree - Unpopping 93

8.2 GTP: Logo 97
8.3 GTP: Consortium Members 98
8.4 GTP: Old and New Logo 115
8.5 GTP: Proto Logos 116
8.6 GTP: CM Business Card 116
8.7 GTP: Webpage - Start Page 130
8.8 GTP: Webpage - News Section 131
8.9 GTP: Webpage - Videos/Demos Section 131
8.10 GTP: Game Developer Ad 132
8.11 GTP: Jungle Rumble 133
8.12 GTP: Have U Seen My Shadow 134
8.13 GTP: Penta-G 135
8.14 GTP: Software Repository Scheme 136
8.15 GTP: Support Forum 137
8.16 GTP: Free Web Advertising 138
8.17 GTP: in profil 12/2005 140
8.18 GTP: GmG Webpage 140
8.19 GTP: GmG Game Voucher 141
8.20 GTP: Resfest Poster 141
8.21 GTP: SIG Membership Agreement 142

Chapter 1

Introduction

1.1 In the beginning

God created the heaven and the earth. And the earth was without form, and void; and darkness was upon the face of the deep. . . . And God said, Let there be light: and there was light. And God saw the light, that it was good: and God divided the light from the darkness.([MosBC],[OAS+11],[PC97])

This thesis also deals with light and the absence of it, when an object blocking its path brings a quick end to the journey of the light ray, giving rise to shadow. It deals with the light that shines in the eye of the computer graphic professional, when he can render more objects, with the same speed and nearly the same quality. It also deals with the light that playing brings to peoples hearts, by touching several, sometimes dark, topics from the realm of computer/videogame-development and -design. And last but not least, it is also about the European Union GameTools Project, the endeavor of carrying the torch of scientific knowledge into the alleged dark realm of the game developer.

1.2 Publications about this work

The author of this thesis has previously published results about it in the following venues:

- The Queried Virtual Shadow Maps algorithm was published at the I3D conference on 3D Graphics and Games in Seattle, WA: Markus Giegl and Michael Wimmer. Queried virtual shadow maps. In Proceedings

of the 2007 Symposium on Interactive 3D Graphics and Games. ACM Press, 2007 ([GW07b]).

- A more practically oriented version of the paper was published in the 5^{th} installment of the excellent ShaderX book series: Markus Giegl and Michael Wimmer. Queried virtual shadow maps. In ShaderX 5 - Advanced Rendering Techniques, pages 249-262. Charles River Media, 2007 ([GW07c]).
- The Fitted Virtual Shadow Maps algorithm was published at the Graphics Interface conference held in Montreal, Canada: Markus Giegl and Michael Wimmer. Fitted virtual shadow maps. In Proceedings of Graphics Interface. CIPS, Canadian Human-Computer Communication Society, A K Peters, 2007 ([GW07a]).
- The Unpopping image space LOD blending technique was published in CGF in 2007: Markus Giegl and Michael Wimmer. Unpopping: Solving the Image-Space Blend Problem for Smooth Discrete LOD Transitions Computer Graphics Forum 26 (1), pages 46-49, March 2007 ([GW07d]).
- An early draft of the basic Unpopping algorithm appeared in the 2^{nd} edition of one of the standard works on 3D computer graphics, “Real-Time Rendering” by Tomas Möller and Eric Haines: Tomas Akenine Möller, Eric Haines. 2002. Real-Time Rendering, Second Edition, pages 391f. A. K. Peters Limited ([MH02]).

The author has furthermore participated in another related research topic, LOD with Near- and Farfield rendering:

- Michael Wimmer, Markus Giegl, and Dieter Schmalstieg. Fast walkthroughs with image caches and ray casting. In Michael Gervautz, Dieter Schmalstieg, and Axel Hildebrand, editors, Virtual Environments 99. Proceedings of the 5th Eurographics Workshop on Virtual Environments, pages 7384. Eurographics, Springer-Verlag Wien, June 1999. ISBN 3-211-83347-1 ([WGS99a]).
- Michael Wimmer, Markus Giegl, and Dieter Schmalstieg. Fast walkthroughs with image caches and ray casting. Computers and Graphics, 23(6):831838, December 1999. ISSN 0097-8493 ([WGS99b]).

Chapter 2

Shadowing

2.1 Introduction

A major part of this thesis deals with creating higher quality shadows in real-time through shadow mapping. This chapter first gives a brief introduction into the problem of shadowing a scene and techniques to do so. It then introduces shadow mapping in more detail, and finishes with a closer look at research in the field shadow mapping. Shadow mapping research is also the main focus of the following chapters, where a new family of shadow mapping algorithms to greatly increase the quality of shadows independent of the view direction is presented.

2.2 Abbreviations

The following abbreviations are used in this chapter:

Abbreviation	Meaning
SM	Shadow Map
SMing	Shadow Mapping

2.3 Shadowing Techniques

Shadows play an important role in (real-time) computer generated images. Apart from applications e.g. in architecture, where one explicitly wants to sim-

ulate the lighting (and therefore also shadowing) of a scene, shadows are of general importance because they:

1. Convey spatial information to the viewer, allowing him to judge e.g. how far above a surface an object is; i.e. it puts objects in relation to one another in space. [1]
2. Increase the realism of a scene by a large amount.[2].
3. Greatly influence the mood of a scene. Unshadowed scenes are usually perceived as being sterile and bland.

Shadowing is the problem of deducing which parts of the scene receive light from a light source, i.e. it is linked strongly to visibility within the scene. A first-order approximation of this problem is the problem of deducing which parts of the scene are visible from a light source, i.e. reducing it to visibility within the scene as seen from the light source; the parts of the scene, which cannot be seen from a light source are then considered to be in shadow. This approximation is called "first hit shadowing" or "direct shadowing", since it does not follow the rays from the light source[3] any further into the scene, but stops at the first surface a ray hits.

This is obviously an approximation, since in reality light is remitted from surfaces, in many cases to a large part within the visible spectrum [4].

This first order approximation of shadowing can also be seen as a subproblem of a general physical lighting simulation where shadows are not treated explicitly, but are a natural result of the general simulation of the behavior of light.

Techniques that follow the light rays emitted from the light source beyond the first hit surface (i.e. go beyond first hit shadowing) are called global illumination techniques. Their goal is to calculate a solution to the "rendering equation", which describes the complete light transport in the scene, and was introduced by Kajiya [Kaj86] in 1986.

[1] For objects whose size is not known to the viewer, this information stemming from shadowing is even more important than in reality, since the objects which are depicted currently neither are rendered with enough surface detail to deduce their size, nor do the output devices on which they are displayed have enough output resolution with regards to the typical viewing distance.

[2] This is even true for non-photorealistic rendering, such as "toon shaded" rendering (i.e. shading characters in the style of animated cartoon characters, with a hard shadow boundary)

[3] Computer graphics uses a ray approximation for the light, which is valid since the objects in the scene typically have dimensions that are much larger than the wavelength of electromagnetic light waves in the visible spectrum, therefore diffraction effects can be ignored.

[4] which is what rendering usually concerns itself with, contrary to e.g. emission in the infrared part of the electromagnetic spectrum.

One global illumination technique is Radiosity [GTGB84]. Radiosity restricts all surfaces to be perfect diffuse reflectors[5]; this leads to a lighting solution which does not depend on the position of the viewer, i.e. which is static for static light sources and a static scene geometry. Radiosity is therefore often used to precalculate static global illumination solutions for eletronic games (see 2.5.4 below).

Basic ray tracing is another approach to approximate a solution to the rendering equation. Basic ray tracing restricts the scene to only contain perfectly reflective and refractive objects. It then follows a ray in object space from the eye-point through each screen pixel into the scene, recursively reflecting/refracting the ray when it hits a surface, until it either hits a light source, or a maximum recursion depth is reached.

If ray tracing stops at the first intersection with a scene surface, and from there only checks if a ray[6] from the intersection point at the surface to the light source intersects scene geometry[7], to discern whether the intersection point (i.e. screen pixel) lies in shadow or not, ray tracing can also be used as a shadowing technique[8].

Stochastic ray tracing ([Kaj86]) can be used to calculate a full solution to the rendering equation which converges statistically. This field is mathematically complex due to its stochastic nature, since care must be taken that the solution does in fact converge. Stochastic ray tracing does not require the restrictions described under radiosity and basic ray tracing above, but can, in principle, handle any type of surface[9] is able to produce physically correct renderings of an arbitrary scene. The price for this is, that it can easily take hours to render one frame for a realistically sized frame buffer[10]. For an overview over stochastic ray tracing algorithms please see Szirmay-Kalos [SK98].

An advantage of ray tracing is, that it can be parallelized well. Basic ray tracing also has advantages when rendering very large models which do not fit completely into main memory.

[5] i.e. the direction into which light is emitted from the surface does not depend on the incoming light direction.

[6] "shadow feeler ray"

[7] i.e. whether the light source can be seen from the intersection point or not.

[8] although currently this approach is very slow

[9] depending on how accurately its surface properties can be measured or modeled mathematically

[10] e.g. 1024×512 pixels

2.4 Shadowing in this Thesis

In this thesis we treat shadowing as an orthogonal problem to the reflective and absorptive nature of the the surfaces of a scene, i.e. we are only concerned with discerning the pixels on the screen which are visible from a light source.

2.5 Shadow Algorithms

The following gives a brief overview over shadowing algorithms for real-time first hit shadowing, from a game developer's perspective.

Following the aim of this thesis, it classifies shadowing algorithms not following strict computer graphics science, but more the way that game developers seek solutions to the problem of real-time shadowing a scene within the restrictions of a respective target hardware.

The order of the presented shadowing techniques is roughly from simple to more complex, corresponding in most cases to a historic ordering with regards to the first use of the shadowing algorithm.

The following terminology is used: A "shadow receiving surface" is a surface which lies at least partially in shadow. A "shadow caster" is an object which blocks light rays, thereby giving rise to shadow receiving surfaces being shadowed (in visibility terminology, the shadow caster occludes at least part of the shadow receiving surface as seen from the light source, or, as visibility is reciprocal, vice versa).

2.5.1 Planar Shadows

Planar shadows (Blinn [Bli88]) use the simplifying assumption that all shadow receiving surfaces can be approximated by a single plane. This shadow receiving plane is then placed "near" the actual shadow receiving surfaces. Evidently this approximation works the better, the more often the shadow receiving surfaces actually coincide with this shadow receiving plane; This is e.g. the case for shadows which fall on a flat floor. This type of shadow is mostly suited for light sources like directional light coming approximately from above, or a point light source hovering above the shadow caster.

Evidently it is not possible to simulate self-shadowing, the shadowing occurring when an object shadows parts of itself, with this technique.

Planar shadows treat each shadow caster in the scene independently of other shadow casters.

Planar Geometric Shadows

Planar geometric shadows render the geometry of the shadow caster projected onto a plane by an additional projection matrix directly in black into the scene. Due to using black for the shadow, overlaps in the projection which occur for concave shadow casters do not pose a problem. Semitransparent shadows[11] from a concave shadow caster (or overlapping shadows coming from different shadow casters) will either display incorrect further darkening in regions that overlap in the projection, or the already shadowed areas have to be protected e.g. by use of the stencil buffer; since this is a very simple shadow approximation technique, this effort usually makes no sense in practice, since systems with graphics cards which support stencil buffers normally are powerful enough to support better shadowing techniques (see below).

An example of good use of planar geometric shadows is the 1999 first-person shooter "Kingpin"[12]. Planar geometric shadows worked well in Kingpin, also due to the relatively planar nature of the levels. The illusion shattered when an opponent would stand above the player on a staircase, and his shadow would extend from his feet into thin air, showing the shortcomings of this simple shadow technology.

Planar Texture Shadows

Planar texture shadows ([MH02], pg 253–260) work like planar geometric shadows, with the only difference that the projected geometry is first rendered opaquely (i.e. with $\alpha = 1$) into the alpha channel of an initially completely transparent texture (i.e. which has been filled with $\alpha = 0$), which is then in turn rendered into the scene as a textured quad, with alpha texturing enabled. The rendering of the geometry of the shadow caster into the texture is done, as to make maximal use of the texture resolution, i.e. so that the geometry touches the sides of the texture.[13]

[11] i.e. shadows which are rendered into the scene semitransparently (instead of opaquely black) to simulate an ambient-light term. In the particular case of planar geometric shadows, the projected geometry is rendered α-blended with the scene.

[12] http://en.wikipedia.org/wiki/Kingpin:_Life_of_Crime

[13] In modern shadow mapping (see Section 2.5.6) terminology, one would say that the rendering is done "focused" on the shadow caster.

Planar texture shadows have the following advantages over planar geometric shadows:

1. Semitransparency for a single shadow caster has no overlapping-concave-parts artifacts, since the shadow caster geometry can be rendered to the texture opaquely, and the textured quad can then be rendered semitransparently into the scene. Overlapping shadows from different shadow casters still pose a problem.
2. For a static relationship between the shadow caster and the light source, the texture can be cached; for complex geometry this makes the application of the shadow to the scene faster, since only one textured quad needs to be rendered[14]
3. If the light source is a directional light, the projected textures can be shared between shadow casters which have the same geometry and orientation in space (e.g. furniture [chairs etc], boxes, columns,...).
4. The shadow edges of the shadow can be antialiased by simply redrawing the projected geometry in antialiased line drawing mode. This gives the shadow a soft shadowy look.
5. The shadow texture can be blurred in image space, again to give the impression of a soft shadow.

2.5.2 Projected Shadows

Projected shadows ([AWG78]) are the older brother of planar shadows, in that they do not need the assumption of only a single shadow receiving plane, but instead actually render a projection of the shadow caster onto the shadow receiving surfaces.

Projected Geometric Shadows

Projected geometric shadows render the geometric projection of the shadow caster onto every shadow receiving surface; for this the projected geometry evidently needs to be clipped to each receiving surface.

Projected geometric shadows have very little practical application, since they have an overall complexity that is the product of the geometric complexity of

[14] What can potentially slow this method down is the need for texture switches, so for simple geometry, planar geometric shadows can be faster.

the shadow caster and the shadow receivers[15]. In addition, they also share the problem of concave shadow casters and overlapping shadows from different shadow casters described under projected planar shadows above.

Projected Texture Shadows

Projected texture shadows ([SKvW+92a]) again treat each shadow caster in the scene independently of other shadow casters. The first step in the algorithm is the same as for planar texture shadows above: The projected shadow caster geometry is rendered opaquely into a texture, only this time the texture is oriented to be orthogonal to the light direction[16]. Then each shadow receiving surface is rendered again, textured with this projected shadow texture, with a texture matrix which projects the texture onto the surface as if the projected shadow texture was a cutout which was illuminated from behind by the light source.

This effectively decouples the geometric complexities of the shadow caster and shadow receivers, leading to a practical algorithm, which shares all the advantages of planar texture shadows (see above) but can create shadows on any kind of shadow receiving surface.

One problem of projected texture shadows is where the light frustum (which is needed to discern which surfaces are in fact shadow receiving surfaces) shall start: If the plane is set too far before the shadow caster (as seen from the light), then surfaces which are not in shadow can be involuntarily be shadowed; if it is too far behind the shadow caster, then surfaces which lie in the shadow will not be shadowed.

Although this does not necessarily pose as big a problem in practice as might seem, it is impossible to find a general solution to it, without adding depth information to the projected texture, effectively arriving at the shadowing technique of shadow maps, described below.

Evidently projected texture shadows can again not supply self-shadowing.

Projected shadows can also be used together with point lights and spot lights. But note that for these types of light sources the size of the projection on a shadow receiving surface becomes the larger the more the distance of the shadow caster to the light source is smaller than the distance to the shadow

[15] . . . thereby revealing the full geometric complexity of the shadowing problem. A large part of shadowing research deals with reducing the complexity of this problem.

[16] The extent of the shadow caster in light space can e.g. be calculated by projecting the bounding box around the shadow caster onto a plane which is orthogonal to the light direction.

receiving surface. When using these light source types together with projected texture shadows one can therefore in practice run into undersampling artifacts, i.e. too little available texture resolution, and therefore the texels of the texture become visible on the shadow receiving surfaces. This is the price one pays for having decoupled the complexity of shadow casters and shadow receivers by discretizing the shadow caster to a texture; see also “shadow mapping” below, where the same problem occurs, and where this thesis presents algorithms to remove or at least reduce the problem.

2.5.3 Blob Shadows

Blob shadows[17] also treat each shadow caster in the scene independently of other shadow casters, and are the smaller brother of the above techniques: They can be combined with either planar shadows or projected shadows, and use the additional simplifying assumption that the silhouette of the projected shadow can be approximated by a circle or ellipsoid (or similar simple geometric shape). This is equivalent to assuming that the shadow caster can be approximated with a sphere or other simple geometric form. Since this approximation is very crude, it is usually only used in conjunction with the assumption that the shadow caster has either a light source moving with it and shining on it directly from above or is lit by a directional light coming from above; the main purpose of this type of shadow, which has been used extensively in eletronic games, is to supply spatial information to the viewer, especially with regards to the distance of the shadow caster to the ground (e.g. when the player avatar jumps).

2.5.4 Lightmaps

Light mapping[18] is a shadowing technology that assigns a distinct texture or part of a texture[19] to each surface in the scene. These textures (“lightmaps”) are calculated offline to carry static information about the shadowing of each

[17] Blob shadows are a simple technique, which, to the authors knowledge, has not been mentioned in any scientific publication or book, so no reference is given; they are mentioned here, since, du to their simplicity, they have been used frequently in eletronic games.

[18] Since in practice the surfaces of the scene are rendered lit, without shadows, and the “lightmaps” are then actually used to modulate (i.e. darken) their respective surface, “shadow maps” would in the author’s opinion be a better name, which is, alas, already taken.

[19] collecting several textures in one larger texture is called a “texture atlas”; usually this is done for performance reasons, to e.g. reduce the number of texture switches, or to use a GPU-enforced quadratic texture efficiently.

surface, which can then be used during rendering to modulate the resulting color of other shaders applied to the surface (diffuse color texture etc), giving the impression of a shadow falling on the surface.

Since calculations are done offline, the lighting simulation used can be arbitrarily complex; the only restriction is that the simulation must treat the surfaces as diffuse, since otherwise the information in the lightmaps could not be static but would depend on the position of the observer. Often radiosity ([GTGB84]) is used to calculate the lighting solution encoded into the lightmap textures, since it produces very convincing results and is inherently based on the assumption of diffuse surfaces; other, faster methods such as ambient occlusion[20],[21] ([Lan02]) are also used.

Lightmaps have been the backbone of static shadowing in eletronic games since the time of id Software's "Quake"[22], due to the fact that they are fast, can be applied efficiently with a fixed function pipeline which supports multitexturing and give nice, moody soft shadows. An interesting fact is that increasing the resolution of the lightmap textures does not automatically lead to better graphics, since very low resolution lightmap textures spread their few texels (smoothly bilinearly interpolated by the graphics hardware) over a large area, so that one cannot discern a clear shadow edge. If the area a texel covers is quite large, an observer will just see a smoothly varying shadow; if, however the texels become smaller, but not small enough (evidently subpixel accurate would be ideal), then an observer will start to pick out distinct texels, and the corresponding jaggy shadow edges, and the perceived visual quality will actually be reduced.[23]

2.5.5 Shadow Volumes

Shadow volumes were first introduced in 1977 [Cro77]. They share characteristics with projected geometric shadows, in that they also project the geometry of the shadow caster onto the shadow receiving surfaces. To reduce the complexity of this process, they do the projection in view space, with the surfaces

[20] http://en.wikipedia.org/wiki/Ambient_occlusion

[21] Due to its better performance, ambient occlusion is a technique that can also be used to update lightmaps in real-time. Note, however, that ambient occlusion, as the name implies, does not produce shadows coming from a light source, but gives only a shadowy impression in the scene, as if it was lit by diffuse light coming from a clouded sky. For dynamically updated global illumination solutions see also section 2.5.7.

[22] http://en.wikipedia.org/wiki/Quake

[23] This is a good example where a seemingly obvious improvement from a technical point of view actually leads to a decrease in perceived quality.

of the scene already rasterized into the depth buffer.

Shadow volumes require the shadow caster to be polygonal. The shadowing operation is done by extending the edges of each polygon of a shadow caster's surface[24] into the frustum formed by shooting rays from the light source[25] through the polygon's vertices, letting the frustum edges start at the polygon vertices. This frustum ("shadow volume") evidently then represents the visibility of the scene with regards to this shadow caster polygon and light source: All points lying within the frustum are in shadow, all points outside of the frustum are not.

The shadowing operation in the depth buffer is then based on the following observation: A point in the scene lies within a shadow volume if a line from a point outside of the shadow volume to the point intersects more front-facing shadow volume faces than back-facing ones. Whether this is the case, can be answered using the scene rasterized into the depth buffer together with the stencil buffer, as follows: For each shadow volume we first rasterize the shadow volume faces which are frontfacing relative to the eye-point into the depth buffer, with depth-compare on and depth-write off. In the stencil buffer, we increase each pixel's counter by one if the pixel on the frontfacing shadow volume surface is visible (i.e. it lies before the nearest pixel in the scene). We then do the same for all backfacing shadow volume faces, decreasing each pixel's counter in the stencil buffer by one if the pixel on the backfacing shadow volume surface would be visible. After this operation, the stencil buffer contains 0 at a pixel, if that pixel lies within the light, and a number > 0 if he lies within the shadow.

A complication occurs for shadow volumes who intersect the front-plane of the view frustum; resolving this can be done by several means, which are beyond the scope of this thesis, and which can for instance be found in [MH02].

An interesting observation is that in games such as "Doom3"[26], which combine shadow volumes with lightmaps (see above), the smoothly varying nature of the lightmaps can clash visually with the completely geometric sharpness of the shadow edges produced by conventional shadow volumes.[27]

One important restriction of shadow volumes[28], as already noted above, is that

[24] As an optimization it actually suffices to do this operation not for all polygon edges, but only for the edges along the boundary between shadow caster polygons which are front-/backfacing relative to the light source, i.e. the silhouette of the shadow caster.

[25] basic shadow volumes work only with point or directional lights

[26] http://en.wikipedia.org/wiki/Doom3

[27] In Doom3 however the relatively low (diffuse surface) texture resolution used clashes even stronger with the sharp shadow volume edges.

[28] Another important drawback is, that one easily gets large overdraw, if many shadow volumes

they only work with shadow casters which are polygonal objects[29]. Neither higher-order surfaces (e.g. NURBS), nor alpha masked objects[30] can be used as shadow casters. Shadow maps (see Section 2.5.6) have no such restrictions.

An advantage of shadow volumes is, that, due to the fact that they work in object space, they produce alias free shadows, something which poses a problem for e.g. shadow maps. This thesis presents algorithms to remove or alleviate this problem when using shadow mapping.

2.5.6 Shadow Maps

Shadow mapping dates back to 1978 [Wil78]. In shadow mapping, the scene is first rendered into a depth buffer, called the "shadow map", from the view of the light. Then, while rasterizing the scene, each pixel is transformed from view space into light space, and the distance (depth) of the current pixel to the light is compared with the entry in the shadow map; if the distance is larger than the shadow map entry, the pixel is shadowed.

Despite its elegance, shadow mapping is still rarely used in eletronic games, due to the prevalence of disturbing undersampling artifacts (see Section 2.6 below).

A recent game to use shadow maps for dynamic shadowing (together with, the author assumes, lightmaps for static shadows) is the role-playing game "Gothic 3"[31]. However, even though the shadow map is not used to shadow the whole scene (giving rise to another type of rather unattractive artifacts where the shadow is simply clipped by a plane at a certain distance), undersampling artifacts become visible, especially when moving the camera closer to the player avatar.

Alhough shadow maps, contrary to projected texture shadows, are usually created for the whole scene, there is nothing prohibiting one to create them on a per-object basis (i.e. focused[32] on an object instead of the whole scene), to e.g. get better shadow resolution (reduced undersampling artifacts) for shadows cast by important objects. The next-generation game engine Unreal 3[33],

overlap in screen space. For a complete list see again [MH02].

[29] For the optimization that, instead of extending every polygon in the shadow caster into a shadow volume only its silhouette is used, its polygons must even form a topologically well-formed polygon mesh.

[30] e.g. leaves on a tree which are commonly represented as one alpha-textured quad per leaf

[31] http://en.wikipedia.org/wiki/Gothic_3

[32] see Section 2.6 or directly [BAS02]

[33] http://en.wikipedia.org/wiki/Unreal_3

recently used in the 3rd person shooter "Gears of War"[34], seems to be using this approach for high-quality shadows cast by dynamic actors. Evidently this approach is better fitted for creating shadows of a small number of dynamic objects in high quality, and would become very expensive if one wanted to shadow a large number of objects, such as a whole forest or an army.

Contrary to shadow volumes, shadow maps can be used with shadow casters which are higher order surfaces (e.g. NURBS) or alpha masked[35], in addition to polygonal objects.

Improving the quality of shadow mapping is one of the main topics of this thesis. Therefore section 2.6 below gives a more in-depth look at the algorithm and previous research in this field.

2.5.7 PRT and Spherical Harmonics Lighting

Precomputed radiance transfer (PRT) and its most famous proponent, spherical harmonics lighting ([KSS02]), are, as the name implies, a precomputation technique, which precalculates a global illumination solution. It then tries to encode this solution efficiently by using a system of basis functions in which it expands the solution and keeps only the dominant coefficients. This allows for fast lookup-retrieval of the global illumination solution for a static scene with a dynamic lighting environment during rendering.

Spherical harmonics lighting was introduced by Kautz *et al* in 2002 [KSS02]. It is based on encoding the light that comes from the outside into the scene into a low frequency environment light map; as the name implies, it uses spherical harmonics as its base functions. The basic spherical harmonics algorithm is not designed to allow for light sources within the scene.

Note that, since the global illumination solution is precalculated and then stored compressed for a specific scene geometry, it is only dynamic in relation to the light, not in relation to shadow casters and receivers in the scene.

The author sees several problems with the idea of precomputing and encoding a global illumination solution, including first hit shadows: The approach means that the whole visibility information of the scene from every point would need to be stored with a high enough spatial resolution. The visibility of a typical, moderately complex scene, say in eletronic games, can be become quite complex very fast; this is due to the the discontinuous nature of visibility. Complexity

[34] http://en.wikipedia.org/wiki/Gears_of_War

[35] e.g. a fence or leaves on a tree, which are commonly represented as alpha textured quads for performance reasons

means that a large amount of information needs to be stored. However any lossy compression scheme (such as expanding into a function system such as spherical harmonics and keeping only leading coefficients) can only cleverly remove redundancies and drop "unimportant" information. If the information needed to restore the visibility information well enough to reproduce high frequency components in shadows (i.e. relatively sharp shadow edges) is complex, then any compression scheme can only go so far, before the resulting shadows will become unnaturally soft.

In practice, trying to use only spherical harmonics lighting to create shadows in a scene, suffers from exactly this problem: For a realistically sized scene[36], even though the amount of data needed by the algorithm is large, it still gives only extremely soft shadows[37], which do not exhibit, at least in the author's eyes, the characteristics that makes them shadows cast by the objects in the scene. Note that this is not the fault of the Spherical harmonics algorithm, which explicitly states that it is for low frequency lighting, but is due to a widespread misconception about what the algorithm is for, respectively what it can deliver in practice.

In practice, the author believes that hybrid approaches, where the direct shadow is computed with non-precomputation techniques, such as shadow volumes or shadow maps, and only the indirect illumination comes from e.g. spherical harmonics is an approach more suited to practical application.

Having said that, however there certainly are applications where a very soft, low frequency dynamic global illumination solution is all one could need; a possible example would be the illumination within the leaves of a tree.

The first-person shooter "Halo 2"[38] tried to use spherical harmonics lighting on a bigger scale, but ended up only used it in very few areas.

2.5.8 Beyond Spherical Harmonics

Bringing practical real-time global illumination to computer graphics is a very active field of research, which is beyond the scope of this thesis. It is also still unclear which algorithms might be developed to be practical enough to see widespread use in eletronic games.

[36] for very small scenes, as always in computer graphics, spherical harmonics can produce impressive results.

[37] In a way this is ironic, since in other shadowing techniques, achieving a soft shadow look is usually what one aspires for; so, in a way, spherical harmonics come from the other side of the spectrum. See also the comment on hybrid techniques below.

[38] http://en.wikipedia.org/wiki/Halo_2

So the following are just two examples of algorithms which the author thinks have practical potential and surpass spherical harmonics in its features.

"Precomputed light paths" is a hybrid algorithm combining conventional first-hit shadowing with additional ray-bounces using precomputed paths of the light through the scene stored in extended lightmaps. Precomputed light paths has been developed within the GameTools Project[39] by Szécsi *et al* [SSKS06]. In contrast to spherical harmonics lighting, precomputed light paths do not require the assumption of a far away light source, but allow for the light source to be within the scene. One thereby arrives at dynamic (with regards to the light) soft indirect illumination effects in addition to direct shadowing. High-quality direct shadowing in algorithms such as [SSKS06] can e.g. be supplied by the algorithms described in this thesis.

As another example, Lightsprint has developed a practically oriented real-time global illumination technology with their "Lightsprint Vision" library[40], which uses a delayed update of the global illumination solution spread over several frames to allow for a dynamic scene while still giving real-time performance.

Note that while allowing for the light source *and* the scene to be fully dynamic is evidently the ultimate goal, allowing for dynamic light sources has in practice much more impact on the possibilities in game design (think e.g. of the effect of the player or player opponents carrying light sources); also, in most eletronic games the scene actually does not actually lend itself to be as dynamic as one would expect, since often large parts of it are architecture or trees/plants; the former is evidently static by nature, while the small movement a tree or a plant makes will have negligible influence on the *indirect* illumination of a scene. Naturally dynamic objects in a scene, on the other hand, such as non-player characters or (in some cases) chairs, etc, due to their small size, have only small influence on the indirect illumination, and the player will most likely not notice the difference.[41] It is therefore the author's opinion that having a fast and stable indirect illumination of the static part of the scene together with direct shadowing from dynamic objects will in many practical cases be preferable to a fully dynamic solution for which e.g. shadows lag behind or the quality of direct shadowing is reduced.

[39] http://www.gametools.org/

[40] http://lightsprint.com/

[41] It is evidently the other way around for the influence of these objects on direct illumination/shadows.

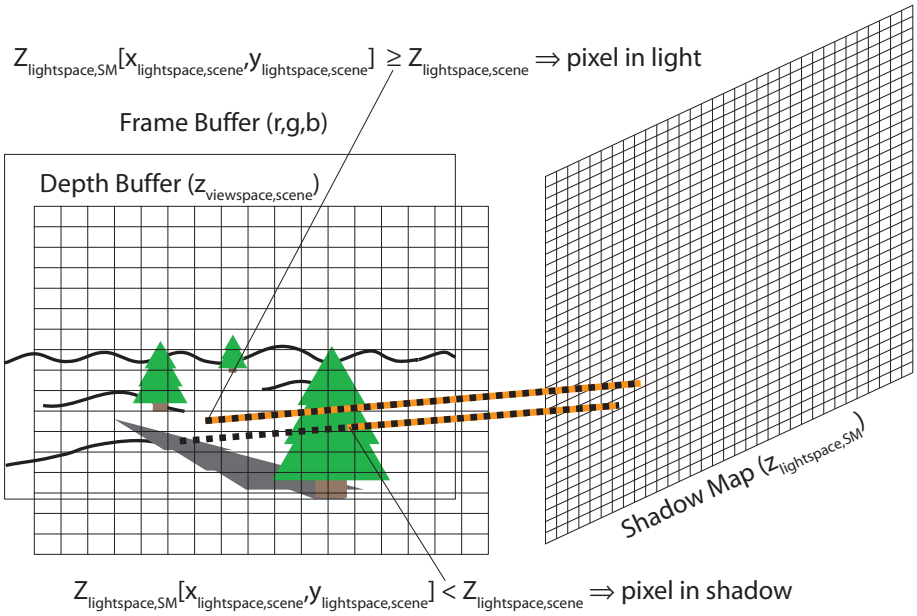


Figure 2.1: Illustration depicting the principle idea behind shadowing a complete scene with shadow mapping corresponding to a directional light source. One can see that the problem is being discretized. Note that the seemingly large resolution of the shadow map on the right is misleading: Even though shadow maps typically have larger resolution than the frame buffer, due to their resolution being stretched over the whole scene, undersampling artifacts become visible even for relatively small scenes.

2.5.9 Further Reading

An excellent overview of classical shadow algorithms in general can be found in Möller and Haines' Real-Time Rendering book [MH02], as well as in [HLHS03] (especially for soft shadow algorithms).

Steiner [Ste06] gives an up-to-date overview over shadowing techniques in general and shadow volumes in particular, whereas Scherzer [Sch05] does the same for shadow mapping.

2.6 A Closer Look at Shadow Mapping

Shadow mapping [Wil78] is a very appealing approach to employ rasterization to solve the first hit visibility problem and use this result to calculate the direct light shadowing of a scene.

Figure 2.1 depicts the principal shadow mapping algorithm:

1. First the scene is rendered into a depth buffer, called the shadow map, from the viewpoint of the light. [42]
2. Then the scene is rasterized into the frame buffer; at each pixel:
   (a) The position of the pixel center is transformed into light space.
   (b) Then the depth entry corresponding to the pixel center in light space is looked up in the shadow map.
   (c) Finally the z-coordinate of pixel center position in light space is compared with the looked-up depth entry from the shadow map: If the depth entry from the shadow map is smaller than the z-coordinate of the pixel in light space, then it is assumed (see below) that the light sees a different surface in the scene first [43], and the pixel lies therefore in shadow; otherwise the pixel is assumed to lie in the light.

The term "...then it is *assumed* that the light sees a different surface in the scene first ..." is used intentionally here, because the simple elegance of shadow mapping has one fundamental problem: The shadow map must contain enough information to allow the visibility queries to be answered with subpixel accuracy for a given frame buffer resolution, otherwise artifacts will be visible, due the regular discretization of the visibility problem.

2.6.1 The Shadow Map Aliasing Problem

The problem that the regular grid of the shadow map does not deliver enough resolution along the SM-axes when being sampled during the SMing process is called "**shadow map aliasing**".

If there is not enough information contained in the shadow map, then all any algorithm can do is try to mask these artifacts, e.g. by means of filtering.

Figure 2.2 visualizes the source for the two forms of shadow map aliasing:

Projection aliasing is stronger the more the texel in the shadow map is projected onto a large area of the shadow receiving surface; this is the case the more the surface normal is perpendicular to the light direction.

Perspective aliasing is stronger the nearer the shadow receiving surface is to the eye-point; this is because due to the nature of the perspective projection,

[42] In the original implementation the light frustum, i.e. the extent of the light in world space was of fixed size. In 2002 Brabec et al. [BAS02] introduced the concept of shadow map focusing, where the light frustum is recalculated each frame as to give a tight fit around the intersection of the view frustum with the scene, to make better use of the shadow map resolution. Unfortunately the focusing at the same time introduces temporal aliasing artifacts.

[43] A ray from the lightsource to the the pixel has its first collision with a different surface.

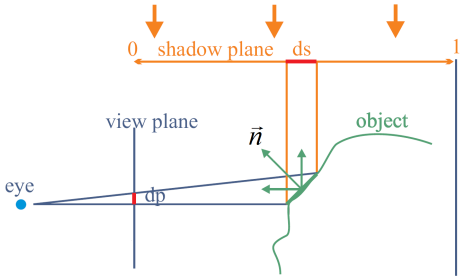


Figure 2.2: Visualization of projection and perspective aliasing.

the closer an object is to the eye-point the more pixels it occupies on the screen. Uniform shadow mapping does not take this into account, but uses the same density of entries everywhere, whereas shadow map reparametrization techniques (see Section 2.7 below) globally warp the geometry that is rendered into the shadow map, so that there will be more information available near the eye point when querying the shadow map; since the transformation is global, they can however do nothing against the local effect of projection aliasing.

Aila and Laine [AL04] and Johnson et al [JLBM05] have proposed algorithms to bypass the aliasing problem by dropping the regular rasterization in the shadow map, and instead creating a shadow map which contains depth entries at the exact locations where the shadow map queries in this frame will occur. These extensions depend on hardware extensions for real-time performance which are not currently available; it is also as of yet unclear how fast the creation of these "irregular shadow maps" would be with an optimized hardware implementation, as compared to "adaptive regular discretization" shadow map algorithms, like the ones presented in this thesis, which are designed to run on conventional graphics hardware.

An obvious way to increase the information contained in a uniform shadow map is to increase the resolution of the shadow map texture. However this becomes impractical very fast, due to its quadratic[44] increase in memory consumption, and also GPU limitations on maximum texture size. For example a 65536^2-float-SM, the approximate size required to shadow a small-to-medium sized scene with subpixel accuracy, would require 16GB of memory. Note that 16GB is not only much more memory than is typically available for the CPU, much less the GPU, but that the fillrate requirements would also be enormous, even if one could store a SM of that size. On current hardware the maximum supported textures size, typically 4096×4096, is the limiting factor, before running out of

[44] See also section 3.6 (pg 30) for a discussion that the required increase is even larger from a perceptive point of view.

available GPU memory.

2.6.2 The Shadow Map Biasing Problem

The resolution of the depth entries in the SM (i.e. along the direction of the light rays) is also a potential source of shadowing artifacts: If the resolution is too low, then even with enough resolution along the direction of the SM-axes artifacts will become visible, since the shadow mapping algorithm will not be able to discern between a shadowed and a shadow casting surface, if they are too close together in light space along the direction of the light rays.

The problem that the resolution in the direction of the light rays is too small, is called the biasing problem, because one can try to hide it by adding a bias (i.e. offset) to the depth entries in the SM. Note, however, that using SM biasing in the end only moves the problem area around, trying to cover up the fact that there is not enough available information.

The problem can also be reduced by clamping all depth entries to the maximum extent of the view frustum along the direction of the light rays, before writing them normalized to the range $[0, 1]$ into the SM.[45]

The problem of too little resolution in the SM depth entries has become less important in recent times due to the emergence of support for textures with $float$ entries.

This thesis deals with the much more problematic area of having too little resolution along the SM-axes, and assumes that the resolution along the direction of the light rays is hight enough, due to the use of linear $float$-SM depth entries. Should the need arise, the techniques presented in this thesis can be combined with any SM biasing technique (see Section 2.7 below).

2.7 Previous Work about Shadow Mapping

Shadow mapping was first introduced by Williams [Wil78] in 1978, whereas Segal *et al* [SKvW+92b] in 1992 supplied the first implementation using standard graphics hardware.

Many of the existing shadow map publications try to solve the problem of aliasing artifacts. Percentage closer filtering [RSC87] alleviates reprojection problems by multisampling the shadow map and Second depth shadow mapping [WM94] can be used to reduce problems due to depth quantization and

[45] For instance under OpenGL the `GL_depth_clamping_NV` extension supports this operation.

self occlusions. Donnelly and Lauritzen [DL06] also store the variance of the scene depth in the SM, to allow for better SM filtering[46]

A number of papers have tried to solve the *perspective aliasing* coming from the perspective view frustum projection through *shadow map reparametrization*. Originally pioneered by Stamminger and Drettakis [SD02], who try to remove perspective aliasing by subjecting the shadow map to the same perspective transform as the viewer, this idea was later refined by Martin and Tan [MT04] with Trapezoidal Shadow Maps, Wimmer et al [WSP04] with Light Space Perspective Shadow Maps and Chong et al [CG04] with A Lixel for Every Pixel. However, all shadow map reparametrization methods deal only with perspective aliasing. They cannot increase the principal resolution of shadow maps, which would be necessary for example to improve projection aliasing, or in cases where the scene is simply too large for the SM resolution, even if the shadow map resolution is optimally (or near optimally) used. Furthermore, they work well only for the case that light and view direction are orthogonal. If these directions are parallel, they have to revert to uniform shadow mapping because the shadow map parametrization runs across the whole screen, not from near to distant points.[47]

Lloyd *et al* [LTYM06] have studied the use of more than one shadow map applied to the sides or slices of the view frustum together with reparametrization techniques intensively, giving more insight into the underlying mathematical structure of the problem. In Cascaded Shadow Maps, Engel [Eng07] presents practical aspects of implementing a view frustum-slicing (see [LTYM06]) SMing algorithm .

Another approach to solve the aliasing problem are adaptive shadow maps [FFBG01], where shadow maps are stored in a hierarchical fashion in order to provide more resolution where it is required due to different aliasing artifacts. However, the approach requires multiple readbacks and does not map well to current graphics hardware. Lefohn [LSK+05] has proposed an extension that makes better use of the GPU, but still performance can be slow. Arvo [Arv04] slices the light view to increase the resolution of the SM.

Tadamura *et al* [TQJN99] partition in z-direction for illuminating landscapes with sunlight, Kozlov [Koz04] uses a cube map in post projective space for SMing.

Brabec et al. [BAS02] improve uniform shadow map quality by focusing the

[46] it seems, however, that this approach only works well for scenes with small depth variance.

[47] Another way to put this, is to say that in this case the viewer sees the shadow map texels in the same size (there is no perspective shortening), so no global reparametrization can use the texels more efficiently, because there is no global imbalance in required shadow map resolution (the resolution would need to be high everywhere).

shadow map to the intersection of the view frustum with the scene.

Note, however, that shadow map focusing, when used without techniques to guarantee subpixel accuracy in the resulting shadow (as presented in this thesis) introduce (dynamic) artifacts of their own: Shadow map focusing makes the effectively available shadow map resolution depend on the view direction and position of the camera in the scene; this leads to a dynamically changing resolution (quality) of the resulting shadow, effectively leading to non-static shadows[48] even in an otherwise completely static scene[49].

A similar problem occurs when using shadow map reparametrization techniques (see above), where the effective shadow map resolution also depends on the view direction, leading to dynamically changing shadow quality, and therefore artifacts in the form of non-static shadows.

Several publications have proposed hybrid approaches, which combine SMing with other shadowing techniques, such as shadow volumes: McCool [McC00] use a SM to construct shadow volumes from it. Sen *et al* [SCH03] store additional geometric information in the SM to be able to create a piecewise linear approximation of the shadow silhouette in the pixel shader. Govindaraju *et al* [GLY+03] use shadow-polygons together with SMs to shadow a static scene with a dynamic light source. Chan and Durand [CD04] use shadow volumes at the shadow edges, which are discerned from a SM.

Note that most hybrid approaches assume a polygonal nature of the shadow caster geometry, forfeiting the advantage that SMs work with scenes containing high order surfaces or alpha masked textures.

[48] To be precise, non-static shadow edges, since the shadow edges are the place where the changing resolution of the shadow map becomes apparent. But any part of a shadow that would be static in reality is perceived as making the shadow non-static, and therefore incorrect.

[49] The change in shadow quality is alas also very noticeable even if the scene is not static, i.e. if the light source and/or the shadow casters move.

Chapter 3

Virtual Shadow Maps

3.1 Introduction

This chapter presents the basis for the following chapters, by introducing a method which allows the increase of the effective resolution of a shadow map applied to a scene far beyond the maximum texture size and (GPU) memory limit, in a brute force manner. It does so by slicing the shadow map into a regular grid of shadow map tiles. For each shadow map tile a shadow map texture is then created on the fly and used to shadow the part of the scene that lies within the light frustum of the tile; the shadow map texture is reused for each tile, i.e. its contents are discarded immediately after it has been used to shadow the scene.

The slicing of the shadow map into tiles overcomes the maximum GPU supported texture size limitation[1], whereas the reuse of the shadow map texture overcomes the physical GPU memory limitation[2]

To make the shadowing of the scene with the shadow map tiles fast, deferred shadowing is introduced, which uses a linear view space depth buffer to shadow the scene, avoiding the need to rerender it for each shadow map tile.

The following chapters explore two much faster, non-brute approaches expanding the principal concept to smarter, adaptive algorithms.

[1] typically around 4096^2 on current GPUs

[2] E.g. to store a complete 65536^2-float-SM, the approximate size required to high-quality shadow map shadow a small-to-medium sized scene, would require 16GB of GPU memory.

3.2 Abbreviations

The following abbreviations are used in this chapter:

Abbreviation	Meaning
LiSPSM	Lightspace Perspective Shadow Maps [WSP04]
SM	Shadow Map
SMing	Shadow Mapping
SM-tile	Shadow Map Tile (see Section 3.3)
SM	Shadow Map
VTSMs	Virtual Tiled Shadow Maps
VTSMing	Virtual Tiled Shadow Mapping

3.3 Virtual Tiled Shadow Mapping

"**Virtual Tiled Shadow Mapping**"[3] ("**VTSMing**") is a brute-force approach for increasing the resolution of the shadow map beyond the maximum texture size supported by the hardware.

Figure 3.1 illustrates the basic VTSMing algorithm, which works as follows.

1. Allocate the biggest shadow map texture supported by the GPU. For example 4096×4096[4].
2. Partition the shadow map along the shadow map x- and y-axis into $n \times n$ (e.g. 16×16) equally-sized tiles (each tile using the full shadow map texture resolution of e.g. 4096×4096 texels, i.e. the effective resolution of the full shadow map in this example is $(16 * 4096) \times (16 * 4096) = 65536 \times 65536$).

   For each tile

   (a) Render a shadow map into the shadow map texture (using culling to the light frustum of the tile and overwriting the shadow map for the previous tile).

[3] The term "virtual" here refers the fact that at no time there exists a complete shadow map in memory, which is of the size with which the scene is effectively shadowed by the algorithm with; it is also inspired by virtual computer memory, which also allows for applications to use more memory than is actually physically present (Note that this analogy is however not complete, since virtual computer memory uses a far slower medium, usually a hard disk, to swap unused memory pages to, whereas Virtual Shadow Maps reuse the same fast memory repeatedly to bypass memory limitations.).

[4] short: 4096^2

(b) Use it immediately to shadow (modulate) the part of the scene which is covered by the current shadow map tile.

3.3.1 Creating the Shadow Map Tiles

To create the n^2 SM-tiles, for each the of the tiles the part of the scene geometry, which lies within the light frustum of each SM-tile needs to be rendered.

An efficient way to do this would be to collect the geometry which is visible within the light frustum of the whole SM in the first pass[5]. In addition, the geometry could also already be culled vs the smaller light frusta[6] of the SM-tiles and collected at each SM-tile. Each SM-tile could then be created with the minimal amount of effort.

Evidently it depends on the implementation effort one wants to invest and the efficiency of the rendering of the geometry lying within a SM-tile light frustum whether this optimization makes sense.

Note however that employing visibility culling in the SM-pass can make sense even for normal SMing, especially if the overdraw in the SM is high; This is e.g. the case when light from the setting sun shines onto a forest scene, leading to a potentially huge overdraw, due to the light direction running alongside the directions of maximum extent of the forest.

As described in the VTSMing algorithm overview above, in each pass only those fragments which lie in the light volume of the current shadow-map tile should be shadowed. Depending on the hardware it can be beneficial to avoid branching in the pixel shader. This can be solved by using a one-pixel border containing the largest z-value around each shadow-map tile. Coordinates of fragments lying outside of the current tile are clamped to this border and therefore never shadowed.

There are two ways to implement the loop over the tiles: multi-pass shadowing and deferred shadowing.

3.3.2 Multi-Pass Shadowing

One way to apply successive shadow map tiles to the scene is by multi-pass rendering. In the first pass, the scene is rendered normally (with full shading

[5] This can be sped up by employing on-the-fly visibility culling, e.g. CHC by Bittner *et al* [BWPP04].

[6] this could be done in projected light space, i.e. in 2D

and depth-writes enabled), with the first shadow map tile applied to it. For each subsequent shadow map tile, the scene is rendered again, but only shadow mapping using the relevant tile is applied to the frame buffer. Pixels falling outside the shadow map tile are suppressed. Depth writes and shading are disabled and the depth comparison function is set to `EQUAL` in those passes (depending on driver support, it can make sense to substitute `LESSEQUAL` for `EQUAL`).

Multi-pass shadowing, although easy to implement, comes with a significant performance overhead of rendering the pure scene geometry (without materials, effects, etc) n^2 times (i.e. as often as the number of SM-tiles). The next subsection therefore introduces a method to speed the process up considerably.

See however section 5.13 (pg 64) for an extension that can speed up multi-pass shadowing, together with an application to rendering high-quality shadowed semitransparent objects.

If multi-pass shadowing is to be employed in practice, it would make sense to use some kind of optimization along the line of collecting the visible geometry in the first pass (e.g. again by employing an on-the-fly visibility culling algorithm such as CHC [BWPP04]), and then either only culling these objects against the screen space bounding rectangle (if available; see Section 4.6 (pg 44) and 5.8 (pg 60)) of the current SM-tile, or creating an optimized buffer in GPU memory which contains only visible geometry and is ready to render (i.e. already transformed to screen space).

3.3.3 Deferred Shadowing

To speed up the application of the shadow map tiles to the scene, we use a variation of deferred shading we call “deferred shadowing” where the shadowing is done using a linear depth buffer of the scene instead of re-rasterizing the scene geometry, and the information needed to do the next shadowing pass, i.e., the next shadow map tile, is created on the fly between the passes (same as with multi-pass shadowing above). The scene is first rendered to a texture that stores eye-space depth, called the Eye-Space Depth Buffer. Each subsequent tiled shadowing pass can then read this texture and calculate the world-space position of the visible surface at each pixel using the screen coordinates and the depth stored in the Eye-Space Depth Buffer. The world-space position is then shadowed using the shadow map tile as before. Note that storing the unmodified eye-space z-coordinate in the Eye-Space Depth Buffer guarantees that the shadow map lookup produces the same results as if the original

scene objects were used for shadow mapping. This is important because any other method of obtaining the z-value, e.g., using window-space z-coordinates (which are highly non-linear) or a fixed-precision w-buffer (if it were still supported on current hardware) would inevitably lead to image artifacts.

In detail this works as follows:

1. In a first pass, render the scene as described above, but into a 4 component 32bit floating point render target. In the pixel shader, store the unmodified eye-space z-coordinate into the α-component. This component forms the Eye-Space Depth Buffer (however for simplicity, we refer to the whole 4-component target as the Eye-Space Depth Buffer). The color of each pixel in the object when lit by this light (ignoring shadowing) is written to the RGB channels. The rendering is done using a conventional $1/z$-depth buffer attached to the Eye-Space Depth Buffer, used for hidden surface removal[7].

[7] Using the eye-space z-coordinate for hidden surface removal would not be a good idea, since we we would for instance void all the depth-buffering hardware acceleration mechanisms of the GPU, such as hierarchical depth-buffer support.

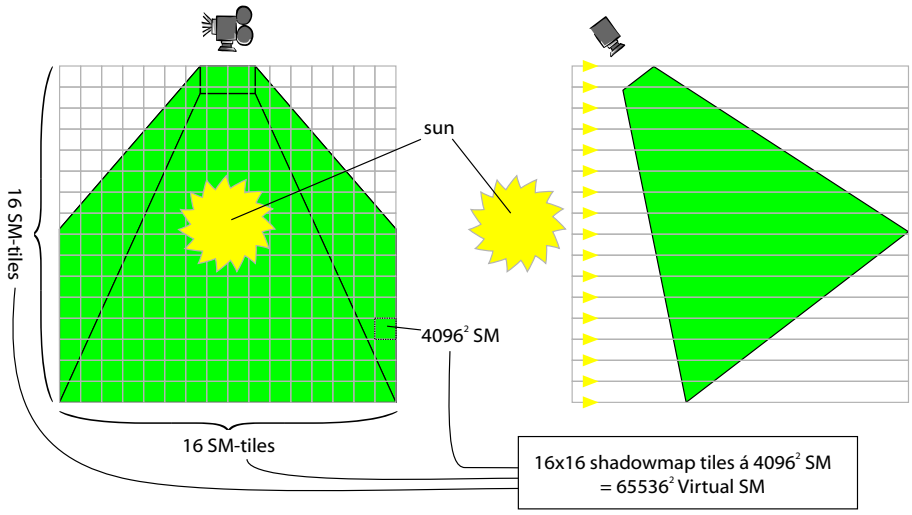


Figure 3.1: Schematic side and top-down view of Virtual Tiled Shadow Mapping: The shadow map (focused on the view frustum) is split into a grid of SM-tiles (here 16×16); on the right the light frusta extend from the light through the SM-tiles (depicted in grey) towards the view frustum (shown in green). Each SM-tile is created on the fly, used to shadow its part of the scene, and then immediately discarded. If each SM-tile uses a 4096^2 SM, then we arrive at an effective SM resolution of 65536×65536 in the example, 256 times larger than using only a single SM.
Note that with normal SMing, only the small "4096^2 SM"-square is used for the whole scene.

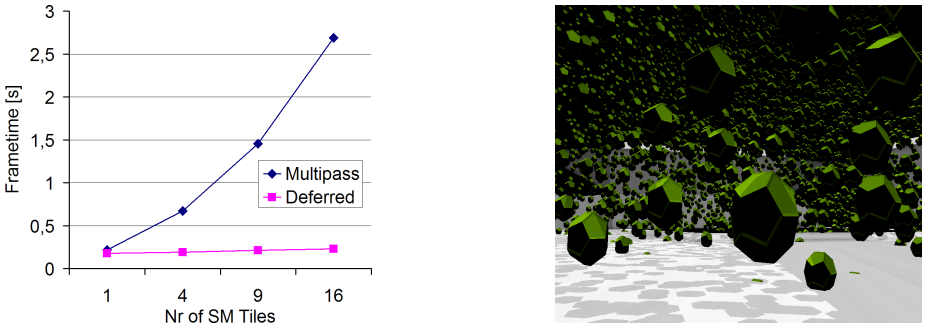


Figure 3.2: Performance comparison of *Multi-pass* and *Deferred* Virtual Tiled Shadow Mapping: 10,000 objs, 1,6 Mtris, 4×4 SM-tiles on a GeForce 6600GT with 256MB, Pentium4 2.4GHz (1GB).

2. For each shadow map tile

   (a) Render a shadow map into the shadow map texture as with multi-pass shadow mapping above.

   (b) Instead of rendering the geometry for the whole scene again, render a full-screen quad with the Eye-Space Depth Buffer bound as a texture.

   (c) In the pixel shader for each fragment, look up the eye-space depth of the fragment in the Eye-Space Depth Buffer's alpha-channel and unproject it into world space (see Formula 3.1 and description below). Using the unprojected fragment, calculate the shadowing term. Then modulate the already-shaded RGB value from the Eye-Space Depth Buffer with the shadowing term.

   (d) The resulting shaded and possibly shadowed fragment is then written to the RGB channel of the Eye-Space Depth Buffer.

3. Transfer the Eye-Space Depth Buffer RGB channels to the frame buffer.

The pixel shader operations in the individual passes are quite straightforward, with the exception of the *unproject operation*. Unlike a standard viewport unprojection, which transforms from window (x_w, y_w, z_w)-coordinates to eyespace (x_e, y_e, z_e)-coordinates, this operation has to deduce eye-space (x_e, y_e, z_e) from (x_w, y_w) (given as texture coordinates, i.e. running from 0 to 1) and z_e. This can be done using the following matrix transform:

$$\begin{pmatrix} x_e \\ y_e \\ z_e \end{pmatrix} = z_e \cdot \begin{pmatrix} \frac{1}{a_x} & 0 & -\frac{b_x}{a_x} \\ 0 & \frac{1}{a_y} & -\frac{b_y}{a_y} \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 2 & 0 & 1 \\ 0 & 2 & -1 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_w \\ y_w \\ 1 \end{pmatrix} \tag{3.1}$$

where the parameters a_x, a_y, b_x, b_y in the first matrix should be taken from the projection matrix P supplied to the graphics API:

$$P = \begin{pmatrix} a_x & 0 & b_x & 0 \\ 0 & a_y & b_y & 0 \\ 0 & 0 & \dots & \dots \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

The second matrix simply transforms from texture-coordinates to screen coordinates, i.e. $[0,1]$ to $[1,-1]$.

see Figure 3.2 for a performance comparison between Multipass and Deferred Shadowing, in a scene containing a large number of objects.

3.4 Shadow Result Texture

The following introduces the concept of the Shadow Result Texture, which can be used to decouple the result of the operation that discerns which pixels are in shadow from the modulation of the scene RGB-colors.

The use of the Shadow Result Texture is a useful extension of Virtual Tiled Shadow Maps, but essential to the algorithm described in the next chapter, Queried Virtual Shadow Maps.

The "**Shadow Result Texture**" is a $1 \times byte$ texture with the same dimensions as the frame buffer, into which we write only the results of the shadowing operation, i.e. whether a pixel is in shadow or not (or, in the case SM-filtering is used, the shadow value in $[0,255]$ which is the result of the SM-filtering operation for this pixel).

Using the Shadow Result Texture allows us to avoid any potential problems with slightly overlapping SM-tiles, since the shadowing results of a tile which is applied later can simply overwrite previous results. It also makes the application of the SM-tiles faster, since we write to a surface with only one byte per pixel. In addition the Shadow Result Texture can also be used to e.g. apply post-processing effects to the shadow, such as screen space blurring depending on distance to the shadow caster.

When the Shadow Result Texture is used with Virtual Tiled Shadow Mapping, we write all the shadowing result values only to the Shadow Result Texture at first. We then modulate the RGB values from the Eye-Space Depth Buffer with

the entries from the Shadow Result Texture in the final step, when transferring the Eye-Space Depth Buffer RGB-color entries to the frame buffer.

3.5 Combination with Shadow Map Filtering

Virtual Tiled Shadow Mapping can be combined with filtering techniques, such as multi- or supersampling, or percentage closer filtering.

The only extension needed to support this is to make the SM-tiles slightly larger. For filter operations such as percentage close filtering, using a filter kernel size of θ shadow map texels, SM-tiles need to overlap by $\theta/2$, so that each tile is large enough to support the filter kernel.

Evidently the performance of using SM-filtering together with Virtual Tiled Shadow Mapping is as good as with normal SMing, i.e. the performance hit on newer cards, which have been optimized for SM-filtering multi-tap access to textures, is very small.

3.6 Problems with Brute Force Refinement

One might think that the fillrate requirements of Tiled Shadow Maps increases with n^2, n again being the number of tiles along each shadow map axis. However, to get a noticeable increase in shadowing quality, one has to effectively double the number of tiles along each axis, which leads to a quadrupling in the number of SM-tiles (i.e. the number of SM-textures that need to be generated each frame). The refinement level n_r, with $n = 2^{n_r}$, is a much better measure of the perceived shadowing quality than the number of SM-tiles n, and the fillrate requirements then become proportional to 4^{n_r}. This means that even for interactive frame times (i.e. of around one second), the achievable maximum virtual shadow map resolution is limited. In addition, for a typical scene, a lot of unnecessary shadow map tiles are generated, due to the brute force nature of the approach.

3.7 Conclusion

This chapter introduced Virtual Shadow Mapping in the form of Virtual Tiled Shadow Mapping. Virtual Tiled Shadow Mapping increases the effective

shadow map resolution beyond hardware limits, in a brute force manner. Deferred Shadowing was introduced as an optimization to increase the speed of Virtual Tiled Shadow Mapping.

The algorithm can be applied to applications that need high-quality shadow map shadows created at interactive rates ($< 1s$; e.g. architectural walkthroughs or the work view of 3D modeling applications[8]; see also below), or very high quality shadow map shadows in a short time frame ($< 1min$; e.g. GPU-based high-resolution shadow map shadowing for e.g. the 3D preview mode of 3D modeling applications or architectural visualizations).

It is easy to implement and can be combined with shadow map filtering techniques to give soft shadowy shadows with a negligible performance hit.

One way to combine the high shadow quality supplied by Virtual Tiled Shadow Maps with a high interactivity is to use a single SM as long as the user moves the camera, and start refining the SM progressively as soon as he stops. We have found that a progression of SM-tiles per axis of $n = 2, 4, 8, \ldots, 2^{n_{r,max}}$, i.e. in powers of 2 up to a maximum refinement $n_{max} = 2^{n_{r,max}}$, is the most efficient tradeoff between reaching a high SM-resolution fast, while at the same time keeping the process visually pleasing.

Note that with this approach, the screen update rate drops to the framerate of the Virtual Tiled Shadow Map rendering at $n = n_{max}$. This means that if there are animated objects in the scene, they might move jerkily when the maximum refinement n_{max} is chosen to be too high. Since in 3D modeling applications or architectural walkthroughs the scene is usually static apart from light movement induced by the user[9], this approach is well suited for these applications.

It is evidently less suited for eletronic games, but could still be used, if n_{max} is not chosen to be too high. See however the next two chapters, which introduce two smarter Virtual Shadow Map algorithms, which adapt to the shadowing needs of the scene, and thereby speed up the shadowing process by an order of magnitude.

[8] e.g. Maya, XSI, 3DSMax or Poser

[9] or very slowly varying on its own, such as in simulated sun movement

Chapter 4

Queried Virtual Shadow Maps

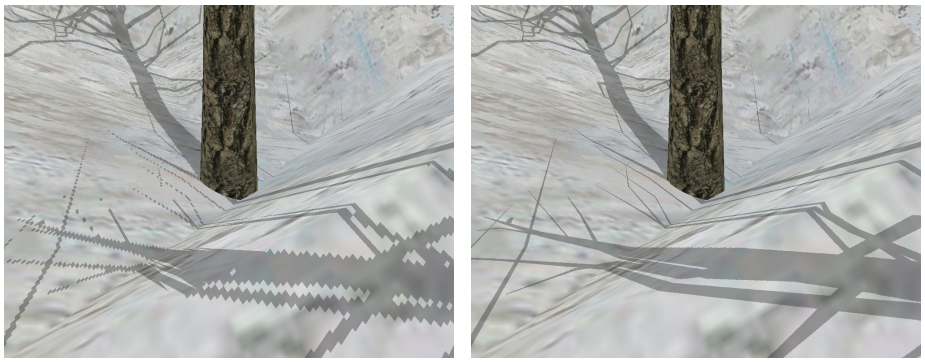


Figure 4.1: Left: shadow map reparametrization techniques (light space perspective shadow mapping is used here) alone cannot guarantee sub-pixel accuracy (leading to perspective aliasing in the lower right corner and projection aliasing on the slope in the middle of the scene), even with a 4096^2 shadow map. Right: Queried Virtual Shadow Maps prevent both types of undersampling artifacts.

4.1 Introduction

In the previous chapter, Virtual Tiled Shadow Maps were introduced, a brute force method to increase the effective shadow map resolution beyond the GPU hardware limit. This chapter presents Queried Virtual Shadow Maps, a smart algorithm which builds on Virtual Tiled Shadow Maps, also increasing the ef-

Figure 4.2: The forest scene used for testing the QVSM algorithm.

fective shadow map resolution available to shadow the scene, while avoiding the quadratic increase in memory consumption. But contrary to Virtual Tiled Shadow Maps, it does so in an adaptive manner, creating more shadow map resolution only where it is needed. It does so without the need to store information from the previous frame, thereby making it suitable for fully dynamic scenes. It can guarantee subpixel accuracy with regards to the shadow map query, getting rid of both projection and perspective aliasing. Queried Virtual Shadow Maps contain an intuitive to use quality vs speed tradeoff parameter, which can be used to tune it to a wide range of graphics hardware.

Queried Virtual Shadow Maps, same as Virtual Tiled Shadow Maps, are orthogonal to and can therefore be combined with other techniques, such as shadow map reparametrization; in the implementation the author used, it was combined with Light Space Perspective Shadow Maps [WSP04].

Figure 4.1 (pg 32) shows a quality comparison between conventional shadow mapping and Queried Virtual Shadow Mapping.

4.2 Abbreviations

The following abbreviations are used in this chapter:

Abbreviation	Meaning
QVSMs	Queried Virtual Shadow Maps
LiSPSM	Lightspace Perspective Shadow Maps [WSP04]
SM	Shadow Map
SMing	Shadow Mapping
SM-Tile	Shadow Map Tile (see Section 3.3)

4.3 Queried Virtual Shadow Mapping

4.3.1 Smart Refinement Preferred

Virtual Tiled SMing is a brute force approach, which makes its practical applicability limited, due to the quadratic increase in the number of SM-tiles which need to be generated to increase the SM resolution by one SM-texture extent.[1]

What we would like to do instead is to adaptively refine the shadow map only where necessary: near the eye-point, and in regions with high projection aliasing. We would like to refine to a high level ($n \geq 16$, i.e. $\geq$ 256 SM-tiles), but do it fast enough so it can be done each frame, and without breaking the GPU friendliness of the algorithm.

One hypothetical way to do this is as follows. Do not shadow the scene directly, but write the results of the shadowing passes into an extra $1 \times float$ texture the size of the frame buffer ("**Shadow Result Texture**", see Section 3.4 (pg 29)).

Then refine the shadow map in quad-tree fashion: First, shadow the whole Shadow Result Texture with a single shadow map tile; then split the tile into 2×2 subtiles, and shadow the Shadow Result Texture with each sub-tile, noting how much the increase of the effective shadow map resolution improves the Shadow Result Texture in each tile. If the improvement achieved by the refinement is small enough, stop processing this tile further. If not, split this tile again into 2×2 subtiles, and so on. Compared to the brute-force approach, this would lead to a greatly reduced number of required shadow map tiles. Unfortunately the GPU is very limited in its ability to execute such "smart" algorithms efficiently, especially those using even moderately complex data structures, such as quadtrees. Therefore we need to move the decision whether to further refine a shadow map tile to the CPU. The only straightforward way to pass the necessary information to the CPU would be reading back the whole Shadow

[1] with declinign return; see Section 3.6

(a) 4096^2 conventional SM

(b) 32×32 2048^2 QVSM

Figure 4.3: Conventional shadow mapping using LiSPSM exhibits undersampling artifacts on the trees in the foreground. The second image was created using Queried Virtual Shadow Maps.

Result Texture after each refinement step and counting the changed pixels, which would be prohibitively expensive.

4.3.2 Queried Refinement: GPU Friendly & Smart

Instead, we have come up with a novel use of the GPU Occlusion Query mechanism, which counts the number of fragments emitted from the pixel shader. Occlusion queries were introduced to support image-space bounding volume visibility tests, and have seen mainstream support by graphics hardware ven-

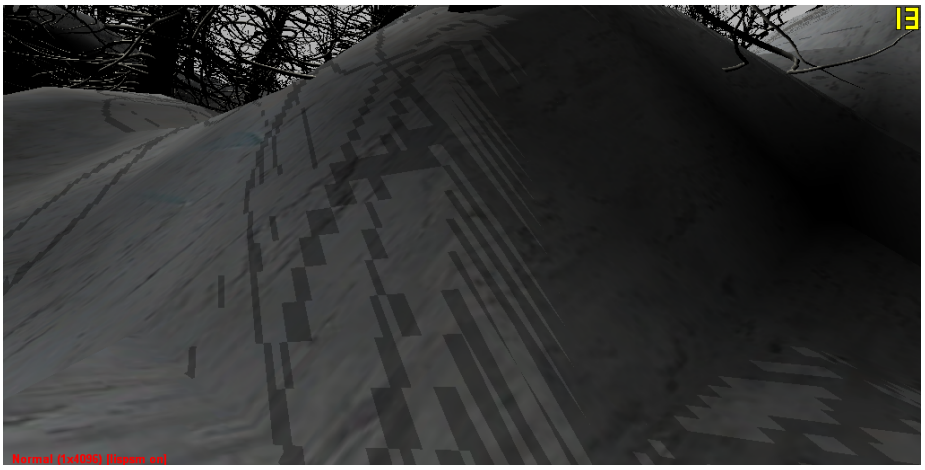
dors for some time now. We use the mechanism for another purpose: when applying a shadow map subtile to the Shadow Result Texture, we instruct the pixel shader to only produce a fragment if the resulting shadow value differs from the previous refinement step (this can easily be done by accessing the previous Shadow Result Texture in the shader). The number of produced fragments η, which is identical to the number of changed pixels in the Shadow Result Texture, is found by bracketing the application of each shadow map tile with an Occlusion Query. The CPU can then decide whether to further refine a tile by comparing the value returned by its corresponding occlusion query with a threshold value η_{min}: if a number of pixels larger than η_{min} have changed, the tile is split into 4 subtiles, otherwise the refinement for this tile stops. In addition we use the maximum number of tiles allowed per shadow map axis, ξ_{max}, as a second refinement termination criterion. Thus, the decision whether or not to refine a shadow map tile can be made without any frame buffer readback, which allows the whole algorithm to produce large effective shadow map resolutions in real time.

4.4 Jump Optimizations

Using the maximum SM texture size supported in hardware for the virtual shadow map texture is, in general, not the best choice. This comes from the fact that the minimum number of Virtual Shadow Maps that need to be filled is $1+4=5$ (the initial shadow map plus one refinement step). The following two optimizations use this observation to speed up rendering by increasing the SM texture size instead of splitting the SM:

4.4.1 Maximum Refinement Jump

This optimization makes use of the maximum tile refinement criterion ξ_{max}, the maximum allowed number of tiles per shadow map axis. Before splitting a tile (of size s), we first test whether the maximum virtual shadow map resolution $\xi_{max} \cdot s$ could also be reached in one step by switching to a larger shadow map texture (i.e., a higher shadow map resolution) instead of splitting the tile. With ξ as the current tile refinement level, we make the jump if $\xi_{\max} \cdot s \leq s_{\max} \cdot \xi$. Since we know that we will reach the maximum user-defined virtual shadow map resolution for this tile and therefore will not refine this tile further, we turn off querying for the shadowing step.



(a) 4096^2 conventional SM

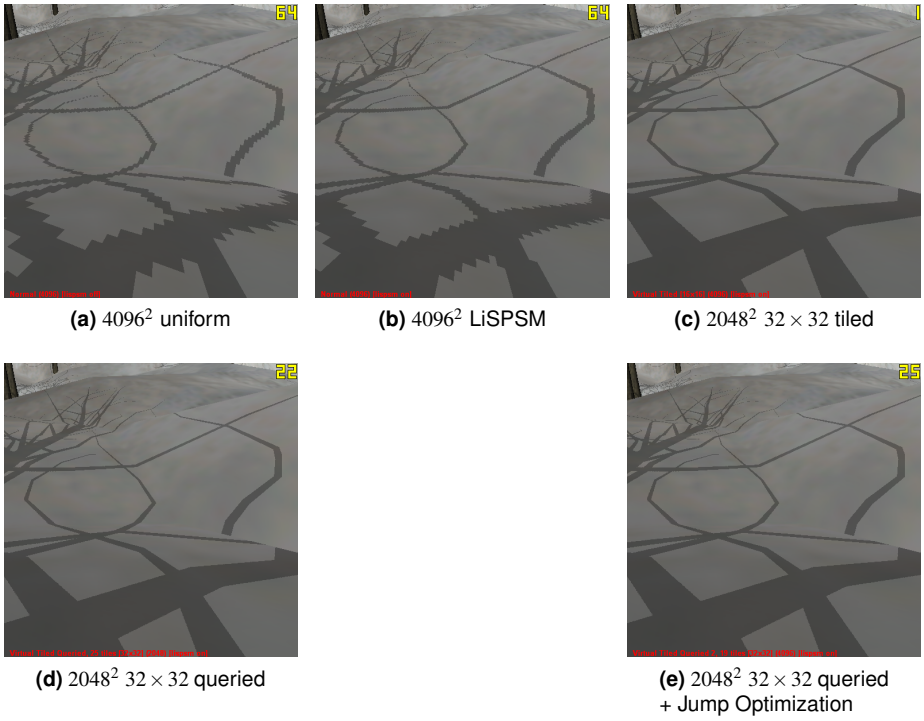
(b) 32×32 2048^2 QVSM

Figure 4.4: Strong projection aliasing (a) is greatly reduced by QVSM (b).

4.4.2 Opportunity Jump

The Opportunity Jump optimization uses a heuristic criterion to predict the future development of η (the number of pixels in the Shadow Result Texture that changed through the last refinement step). If the prediction is that η will become smaller than η_{min} within a "jump distance" (number of refinement steps) $\frac{s_{\max}}{s}$, then we again do not refine the tile, but increase the shadow map texture size instead. We assume that η decreases at least by a factor f_η in each refinement step in the vicinity of η_{min}; f_η is a constant factor, which can be set by the user according to his quality requirements (see Results section below for a discussion of meaningful values for f_η). We make the jump, if $\eta \cdot (f_\eta)^{\frac{s_{\max}}{s}} \leq \eta_{\min}$.

We again turn off querying for the shadowing step and stop any further refinement because, in the unlikely case that the tile does not reach the intended resolution, it would be disproportionally expensive to split the tile further, because we would have to use 4 shadow maps with $s = s_{max}$ (which would be too costly, since the premise is that we do not use $s = s_{max}$ for the SM-tiles from the start because of the cost of generating several $s_{max} \times s_{max}$ shadow maps).



(a) 4096^2 uniform **(b)** 4096^2 LiSPSM **(c)** 2048^2 32×32 tiled

(d) 2048^2 32×32 queried

(e) 2048^2 32×32 queried + Jump Optimization

Figure 4.5: Comparison of the different techniques using a 512×512 frame buffer, 32×32 tiles maximum refinement and a 2048^2 shadow map texture (framerates can be seen in the upper right corner).

4.5 Results

Figure 4.2 (pg 33) depicts the test scene used for the QVSM research. It has a large number of high-frequency shadow casters (branches), which exhibit self shadowing as well as receiving shadows in an irregular manner, by being far from being able to be well approximated by a shadow receiving plane. It also quite naturally leads to the eye-point being potentially very near to a shadow receiver in the form of low hanging branches or the trunk of a tree, therefore making it harder to hide shadow mapping artifacts. In addition the hilly structure

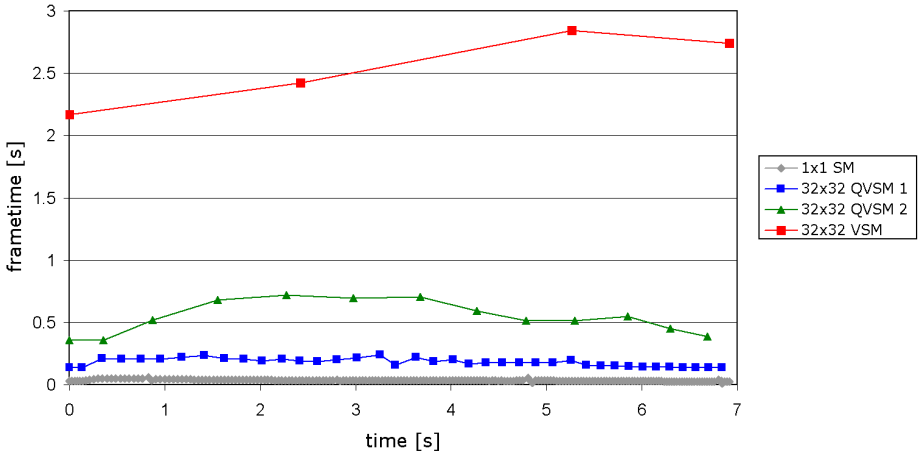


Figure 4.6: Performance comparison between Virtual Tiled and Queried Virtual SMing along path in forest test scene (figure 4.9 (pg 46)).

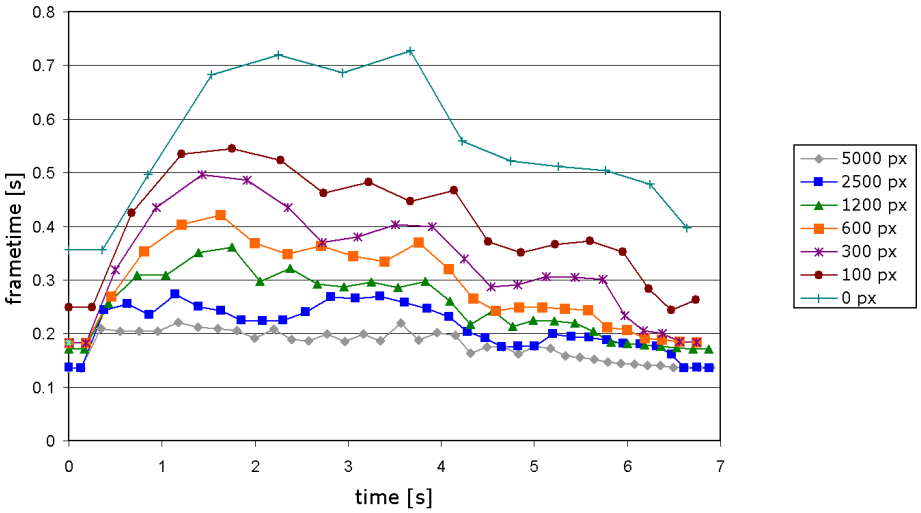


Figure 4.7: Performance comparison along path in forest test scene for several values for QVSM threshold parameter η_{min}.

of the ground gives rise to projection aliasing.

Unless otherwise noted, all results were created on an ATI Radeon X1900XTX with 512 MB of RAM and a Pentium4 3.4 GHz (2 GB RAM).

Figure 4.6 (pg 39) shows frame time curves from a forest scene with 5×10^6 triangles rendered into a 1024×1024 frame buffer. The uppermost curve depicts brute force Virtual Tiled SMing, using 16×16 4096^2 shadow maps; QVSM 1 & QVSM 2 show Queried Virtual SMs with 2048^2 SM-tiles, 32×32 maximum refinement and jump optimizations to 4096^2 using $\eta_{min} = 2500$ and $\eta_{min} = 0$ re-

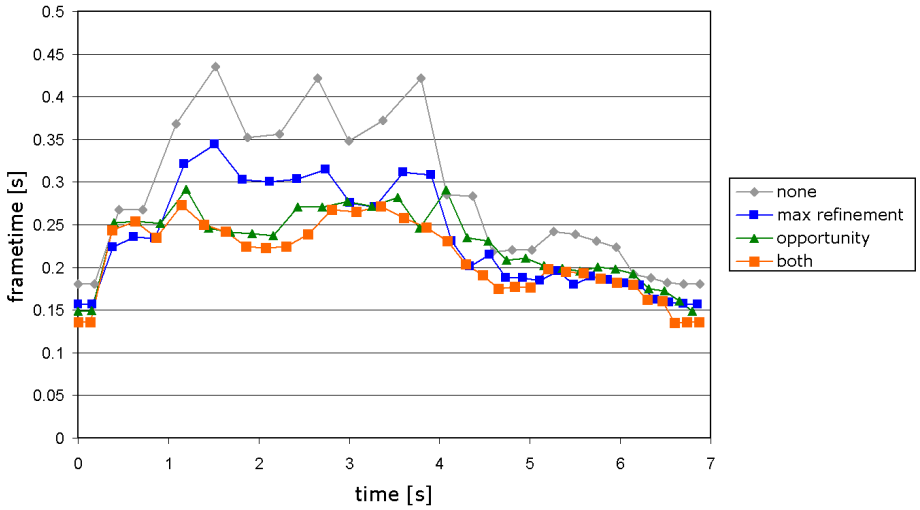


Figure 4.8: Performance comparison along path in forest test scene for QVSM Jump Optimizations.

spectively. The SM-curve finally gives the frame times for conventional SMing using the maximum 4096^2 shadow map texture currently supported in hardware (leading to greatly reduced shadow quality). LiSPSM [WSP04] was active for all SM renderings. One can see that for $\eta_{min} = 0$, i.e. the exact same shadowing quality, Queried Virtual SMs are more than 4 times faster than brute force Virtual Tiled SMing; for $\eta_{min} = 2500$, which still gives excellent shadow quality, Queried Virtual SMs are nearly 15 times faster. The effective Virtual SM resolution used to shadow the scene is $32 \times 2048 = 65536$, as compared to 4096 for conventional SMing. Figure 4.9 (pg 46) shows a screenshot along the path, comparing the visual quality of conventional SMing and QVSMing.

The frame times for several values of η_{min} (the minimal number of pixels that need to change in a SM-tile refinement step for the tile to be refined further) along the same forest path are shown in figure 4.7 (pg 39). One can see that the frame times fluctuate more the smaller η_{min} becomes. This is due to the fact that a smaller η_{min} makes the algorithm more sensitive towards changes in the scene (eye-position, light direction etc), leading to more fluctuation in the number of SM-tiles that need to be created.

Figure 4.8 (pg 40) compares frame times for jump optimizations with $f_\eta = \frac{1}{4}$. One can see that Opportunity Jumping has, in general, the greater effect. What one can also see from the curve is that Opportunity Jumping also has a very beneficial influence on the maximum frame times, in that it cuts the frame time spikes, contributing to a smoother framerate. Having observed the behavior of η in the vicinity of η_{min}, f_η should be chosen to lie between $f_\eta = \frac{1}{2}$ to $\frac{1}{8}$. $f_\eta = \frac{1}{2}$

is a very conservative assumption, while $f_\eta = \frac{1}{8}$ is rather aggressive and can lead to some minor artifacts when a tile does not get refined to the desired quality; $f_\eta = \frac{1}{4}$ in general is a good compromise in practice.

Figure 4.1 (pg 32) shows a comparison between a scene shadowed with a 4096^2 SM using lightspace perspective shadow mapping on the left and Queried Virtual SMing on the right. The screenshot on the left exhibits not only projection aliasing (on the slope in the middle of the screen), against which lightspace perspective shadow mapping cannot help, but also perspective aliasing, due to the fact that the resolution of the shadow map is too small, even with shadow map entry arrangement. QVSMing on the right gives subpixel accuracy and thereby removes both types of aliasing artifacts.

Figure 4.3 (pg 35) shows typical undersampling artifacts on the trees in the foreground, despite using LiSPSM. QVSM (depicted on the right) shadows the scene subpixel accurate, using 19 SM-tiles (17×2048^2 and 2×4096^2; the latter coming from optimization jumps) giving an effective SM resolution of $65,536^2$. Figure 4.4 (pg 37) shows strong projection aliasing on the right due to self shadowing. QVSM on the left removes the projection aliasing nearly completely; note that the high precision of the resulting shadow reveals the nature of the underlying geometry by showing its triangular nature - one can see that the terrain of the scene is much coarser than the trees.

The number of SM-tiles generated by the algorithm depends on the choice of ξ_{max}, η_{min} and $s_{\max}$ and the size of the frame buffer. For quad-splitting of the SM-tiles, the number of SM-tiles generated by the algorithm is $1+4 \cdot k+l_{jump-opt}$, i.e. $\{1,5,9,13,17,\ldots\}+l_{jump-opt}$, where $l_{jump-opt}$ is the number of tiles generated by jump optimizations. In practice we found that for our test scene and frame buffer size, a typical case would be $k = 4$ and $l_{jump-opt} = 2$ leading to 19 SM-tiles. Note that having 6 SM-tiles is a lower bound in practice, since the necessary initial refinement step from 1 to 4 SM-tiles already leads to the generation of 5 SM-tiles overall; if this is followed by a jump optimization refinement, then we arrive at $5+1 = 6$ SM-tiles (having less SM-tiles would mean that brute force refinement into 2×2 SM-tiles would be the better choice). In practice the initial refinement step typically is followed by at least one further refinement step and a jump optimization refinement, leading to ($k = 2$,$l_{jump-opt} = 1$) 10 SM-tiles overall.

Figure 4.5 (pg 38) shows a comparison between different shadow mapping approaches and maximum refinement level ξ_{max}. One can see that sample redistribution methods, represented here by LiSPSM, cannot sufficiently increase the effective shadow map resolution for this view direction.

For the refinement parameter $s_{\max}$, we found that for an NVidia GeForce 6600GT with 256MB of RAM, 1024×1024 shadow map textures together with $s_{\max} = 2048$ proved to be efficient, whereas for an ATI Radeon 1900XTX with 512MB of RAM, using 2048×2048 with $s_{max} = 4096$ proved to be a good choice (both graphics cards support a maximum texture resolution of 4096×4096).

One problem that could arise in practice would be visible SM-tile boundaries in the resulting shadow due to precision issues. We did not observe such problems in our implementation, but should this problem arise, it would be very easy to fix by simply making the SM-tiles overlap by a slight amount. Overlapping the SM-tiles leads to no artifacts, since the shadowing result does not need to be combined with the existing Shadow Result Texture value, but overwrites previous results. In addition higher refined SM-tiles will be generated later than lower refined ones so there is also not even a potential reduction in quality in the small overlap area.

Since shadow mapping and its artifacts (or absence thereof) are best observed in motion, please also go to http://www.cg.tuwien.ac.at/research/vr/vsm, where you can find some demonstration videos.

A non-scientific version of this chapter was published as an article in ShaderX 5 ([GW07c]) from the excellent ShaderX book series; an executable and source-code samples can be found on the CD-ROM accompanying the book.

4.5.1 Extensions and Optimizations

The following lists some further optimizations that can be applied to the algorithm together with some results:

1. Instead of always splitting each SM-tile into 2×2 subtiles (quad-splitting), one can also split it along each SM-axis alternatingly (binary-splitting). Theoretically this would allow the algorithm to better adapt to scenarios where the required SM resolution differs between the two SM-axes. We have implemented this extension and have observed that frame times were, in general, higher than with quad-splitting. An analysis of the created SM tiles shows that the problem is twofold: First, in many cases there is not enough difference in required SM-resolution along each SM-axis; this means that in the end binary-splitting does a quad-split – but it costs $2 + 2 \times 2 = 6$ tile generations (first split the tile into 2 subtiles along one axis, then split the 2 subtiles into 2 sub-subtiles each), instead of just $2 \times 2 = 4$ if we do the quad-split immediately. Second, binary-splitting only gives information about the refinement status along one SM-axis in

each refinement step; so even if this is not necessary the algorithm in many cases has to do "one more split" to make sure that the tile resolution is adequate along both SM-axis. In addition, binary-splitting is harder to combine with jump optimizations efficiently, again due to the fact that in each refinement step only information about a single SM-axis is available. We conclude that the cost of binary-splitting outweighs its benefits for practical applications.

2. Another idea would be to not split into 2×2, but into $n_{sub} \times n_{sub}$ $(n_{sub} = 3,4,\ldots)$ subtiles in each refinement step. We have included this in our algorithm, but, again, this lead to worse frame times in all test scenes. This is because the number of generated SM-tiles becomes extremely large, even if only one tile is refined twice (which is typically the case simply due to perspective aliasing near the eye-point): Even for 3×3 splitting, this already leads to $1 + 2 \times (3 \times 3) = 19$ tiles (the initial tile, a subtile and sub-subtile), compared to 9 tiles $(1 + 2 \times 4)$ for $n_{sub} = 2$.

3. Using the relative metric $\eta_{rel} < \frac{n_{pixels-changed-in-tile}}{n_{pixels-in-tile}}$ instead of the absolute $\eta_{abs} < n_{pixels-changed-in-tile}$ could be a better choice for deciding whether to further refine a shadow map tile. The problem here is to get $n_{pixels-in-tile}$:

   (a) One possible way would be to reapply the shadow map tile to the Eye-Space Depth Buffer (without shadow map lookups, of course), again bracketed with an OcclusionQuery, but always emitting a fragment in the pixel shader if it lies within the current tile. The result of the OcclusionQuery would then be $n_{pixels-in-tile}$. We have tried this and unfortunately the practical cost of this operation was so high that it outweighed any potential benefit. See "Hardware Extensions for Better GPU to CPU Feedback" below for a potential future better way to get η_{rel}.

   (b) A less accurate but faster method would be to calculate the number of screen-space pixels in the projection of the shadow map tile onto the ground plane of the scene (trapezoid clipped to screen-space coordinates), and use this as an approximation for $n_{pixels-in-tile}$. It would of course depend on the characteristics of the scene whether this approximation works well or not.

4. Another approach to deferred shadowing would be to write each point's xyz-coordinates into a float render target (RT), using multiple-RT (MRT) functionality to write the color to a second RT: We chose not to do so, since even modern graphics cards often only support MRTs having the same bit depth; this would have meant that we would have to use two

$4 \times float$ RTs. Since the transformation given under 3.1 (pg 28) achieves the same result with using only one linear depth entry which can be stored conveniently in the alpha-channel of just one $4 \times float$ RT, we chose this approach to implement deferred shadowing.

4.6 Hardware Extensions for Better GPU to CPU Feedback

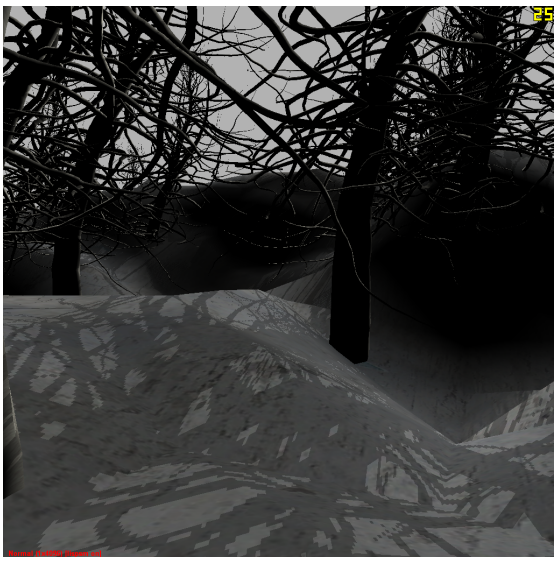
We use the hardware occlusion query mechanism as a counter to efficiently pass back information from the GPU to the CPU. This mechanism is like a tiny loophole, since only one value can be passed back per rendering pass, and increasing the counter is linked to emitting a color value from the pixel shader (which, fortunately, in our case is what we want to do anyway). We are sure that many more smart, adaptive algorithms could be combined with fast GPU rendering if this GPU functionality were extended to include several signed integer registers similar to the occlusion query fragment counter, which could e.g. be incremented/decremented (possibly by an arbitrary amount), and which would support atomic min/max operations. Since these operations are independent of the order of execution, they would be compatible with the highly parallel vector processor design of modern GPUs. With just one additional counter register, one could, for instance, count the total number of pixels corresponding to the current shadow map tile in addition to the number of pixels that have changed in the last refinement step, allowing us to employ different refinement metrics. With 4 additional registers with min/max support one could find the screen-space bounding box around the area influenced by a shadow map tile, reducing the number of pixels that need to be touched when applying a shadow map tile to the scene.

4.7 Conclusion

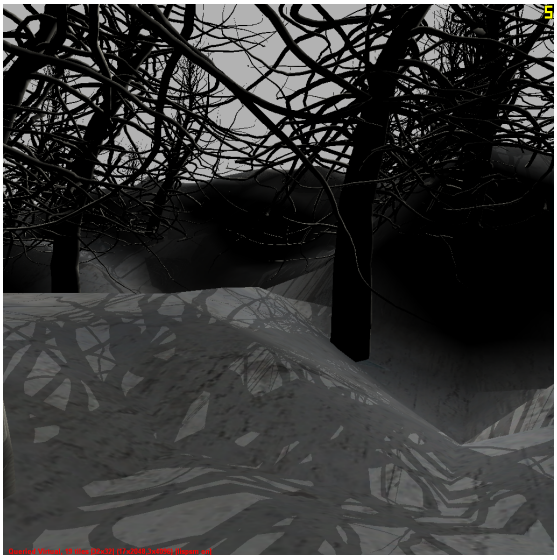
This chapter presented a novel approach to increase the effective resolution of a shadow map without incurring the respective memory cost and bypassing maximum texture size limits, in a manner that is adaptive to the shadowing resolution needs of the scene. Starting with the brute force approach of Virtual Tiled Shadow Maps, we have shown that it can be made faster by an order of magnitude by employing the GPU's occlusion query mechanism to get back

information about the effect of a SM-tile refinement step to the CPU. This information in turn can be used by the CPU to guide the refinement process. The refinement metric is directly correlated to the number of pixels that changed in the scene due to the SM-tile refinement, allowing the algorithm to reduce or even remove perspective *and* projection aliasing.

We have also proposed hardware extensions that should integrate well with existing hardware architectures, which could be used to improve the efficiency of the algorithm even further and allow for other smart algorithms that combine GPU power with CPU versatility.

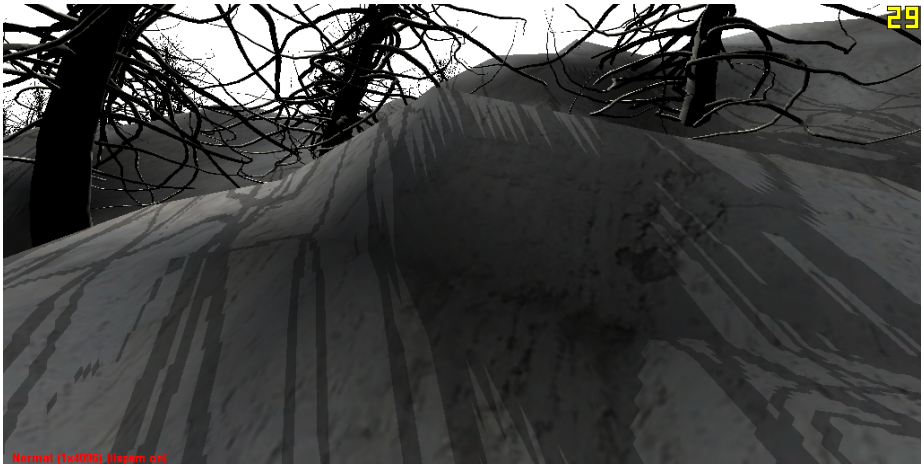


(a) 4096^2 conventional SM

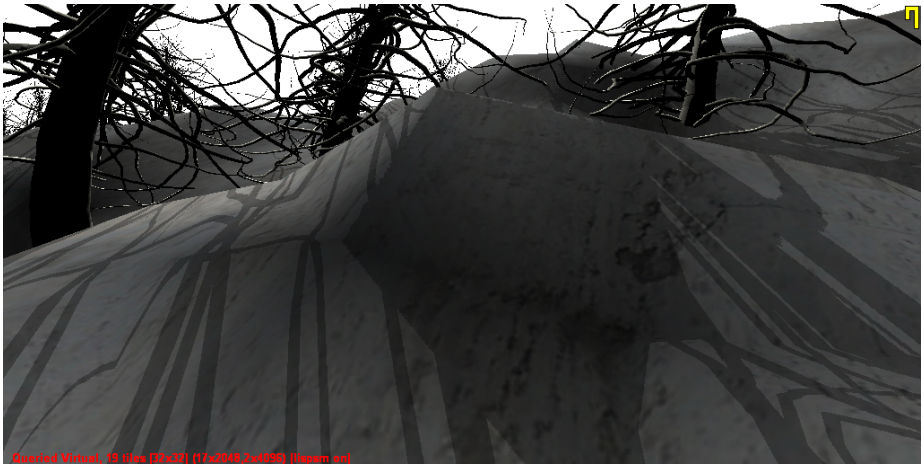


(b) 32×32 2048^2 QVSM

Figure 4.9: Quality comparison in forest test scene, rendered to 1024^2 frame buffer. Screenshot taken along path whose frame times are depicted in figure 4.6 (pg 39). The effective SM resolution applied to the scene on the right is 65536^2, 256 times larger than the SM on the left.



(a) 4096^2 normal shadow map



(b) 32×32 QVSM

Figure 4.10: Comparison of quality and performance of normal SMing with QVSM. Normal SMing exhibits strong projection aliasing; QVSM strongly reduces the aliasing artifacts (thereby also revealing the coarse polygonal nature of the terrain).

4.8 QVSM Core Algorithm

The basic version of Queried Virtual Shadow Maps works as follows (the following is schematic C++ code to facilitate the understanding of the core algorithm, not the actual implementation):

```
const int nr_tile_per_axis_max = 16;
const int nr_pixel_min_changed = 300;

ShadowResultTexture& srt = GetShadowResultTexture();
ShadowResultTexture& srt_old = GetShadowResultTextureOld();

// Set the current shadow result texture to "completely lit"
ClearTexture(srt,1.0);

// Render Scene, including the linear Viewspace Depth into
// (r,g,b,a=depth)-float-Texture with attached
// conventional depth buffer (clear both before rendering)
RenderSceneIntoEyeSpaceDepthBuffer();

// Calculate the light-space matrix
// and focus the whole shadow map
D3DXMATRIX light_space;
CalculateLightSpace(&light_space);

// Queue holding the shadow map tiles to be rendered
SmTileRefinementQueryContainer smtrqc_render;
// Queue holding the shadow map tiles with pending queries
SmTileRefinementQueryContainer smtrqc_pending;

// Prepare the initial SM-tile covering the whole
// SM for rendering
smtrqc_render.AddQuery(
  new SmTileRefinementQueryPixelCount(
    new SmTile(0,0,1) // tile at (0,0), nr_tile_per_axis = 1
  )
);

// Refine & Render SM-Tiles

while(!smtrqc_render.empty() || !smtrqc_pending.empty()) {

  // Copy current shadow result texture to
  // old shadow result texture. The "old" is used
  // as an input to the pixel shader to decide
  // whether or not to reject the new fragment.
  // On a NV GF6600 and ATI X1900, it is faster
  // to not copy and use SRT_old = SRT instead.
  CopyTextureFromTo(srt,srt_old);
```

```
// Shadow the current ShadowResultTexture with all
// pending shadow map tiles
while(
  SmTileRefinementQuery* p_smtrq_render =
    smtrqc_render.PopNextQuery() )
{
  // Create the shadow map texture for the curren tile
  const SmTile& sm_tile = p_smtrq_render->GetTile();
  const ShadowMapTexture& smt =
  RenderSceneInLightFrustrumIntoShadowMap(sm_tile);
  // Shadow shadow result texture with the just created
  // tile-SM, clamped with occlusion queries
  p_smtrq_render->QueryStart();
  ShadowShadowResultTextureWithSmTileShadowMap(
    sm_tile.ProjectionMatrixGet()*light_space, // light frustrum
    smt, // do SM lookups here
    srt, srt_old  // write into srt, compare with srt_old
  );
  p_smtrq_render->QueryEnd();
  // Add query to the pending occlusion query queue
  smtrqc_pending->AddQuery(p_smtrq_render);
}

// Wait until the first occlusion query is ready
while( !smtrqc_pending.FinishedQueryAvailableQ() );

SmTileRefinementQuery* p_smtrq_ready =
  smtrqc_pending.PopNextFinishedQuery();

// Decide whether to split the tile into 2x2 subtiles
if(p_smtrq_ready->NrTilePerAxis() < nr_tile_per_axis_max &&
  p_smtrq_ready->NrPixelChanged() > nr_pixel_min_changed)
{
  SmTile&  sm_tile = p_smtrq_ready->GetTile();
  // the subtiles live in a virtual space 2x
  // as large in each dimension as the parent
  int ix_sub = 2 * sm_tile.TilePosX();
  int iy_sub = 2 * sm_tile.TilePosY();
  int nr_tile_per_axis_sub = 2 * sm_tile.NrTilePerAxisGet();
  for(int ix_rel=0; ix_rel < 2; ++ix_rel) {
    for(int iy_rel=0; iy_rel < 2; ++iy_rel) {
      smtrqc_render.AddQuery(
        new SmTileRefinementQueryPixelCount(
          new SmTile(
            ix_sub + ix_rel,
            iy_sub + iy_rel,
            nr_tile_per_axis_sub
          )
        )
```

```
        );
      }
    }
    DEL(p_smtrq_ready); // this tile query is finished ;-)
  }
}

// Shadow the scene by modulating it with the ShadowResultTexture
// FrameBuffer.RGB = ShadowResultTexture * EyeSpaceDepthBuffer.RGB
ShadowSceneWithShadowResultTexture();
```

Chapter 5

Fitted Virtual Shadow Maps

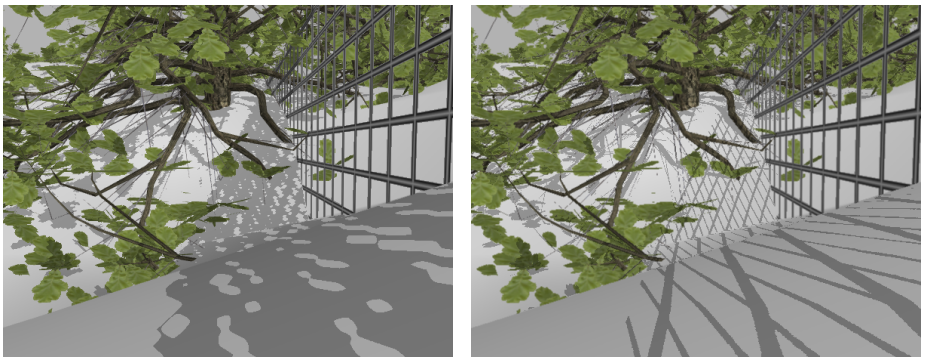


Figure 5.1: Left: Shadow map reparametrization techniques (lightspace perspective shadow maps is used here) alone cannot guarantee subpixel accuracy for neither all light directions nor large scenes, even with the largest shadow map supported by the GPU. Right: Fitted Virtual Shadow Maps allow the scene to be shadowed with subpixel accuracy, while still supplying real-time framerates.

5.1 Introduction

This chapter presents Fitted Virtual Shadow Maps, another smart Virtual Shadow Map based algorithm, able to remove or greatly reduce shadow mapping undersampling artifacts perspective and projection aliasing. Like Queried Virtual Shadow Maps, it also aims to refine the shadow map only where needed, and is also designed to be fast enough to do the full refinement each frame.

Contrary to Queried Virtual Shadow Mapping, instead of counting the number of changed shadow pixels in the last refinement step, and use this metric to decide whether to further refine a SM-tile into 4 sub-tiles, Fitted Virtual Shadow Mapping tries to discern beforehand what SM-resolution is needed where in the scene.

It does so by first sampling the scene from the eye-point on the GPU to be able to calculate the needed shadow map resolution in different parts of the scene. It then processes the resulting data on the CPU in several steps, finally arriving at a hierarchical grid structure, which is then traversed in kd-tree fashion, shadowing the scene with shadow map tiles of the required resolution where needed. Fitted Virtual Shadow Maps allow for shadow quality to be traded for speed through an intuitive to use parameter, which allows for a homogenous quality reduction in the whole scene, down to the quality of normal shadow mapping.

Figure 5.1 compares the quality of conventional shadow mappingwith Fitted Virtual Shadow Maps.

5.2 Abbreviations

The following abbreviations are used in this chapter:

Abbreviation	Meaning
FVSMs	Fitted Virtual Shadow Maps
LiSPSM	Lightspace Perspective Shadow Maps [WSP04]
SM	Shadow Map
SMing	Shadow Mapping
SM-Tile	Shadow Map Tile (see Section 3.3)
SMTMM	Shadow Map Tile Mapping Map (see Section 5.3)

5.3 Fitted Virtual Shadow Mapping

FVSMs works by discerning each frame what SM-resolution is needed where in the scene. The following gives an overview over the algorithm (see the following sections for details): (In the following, ”supplying subpixel accuracy” for a SM-tile means that for the current eye-point and frame buffer resolution, the SM-tile has enough resolution in its corresponding SM texture, so that when the scene

(a) 4096^2 normal SM

(b) FVSM 32×32 max SM-tiles

Figure 5.2: Quality comparison at end of test path through winter forest (LiSPSM SM reparametrization active in all screenshots).

is shadowed with the the SM-tile, the resulting shadow is subpixel accurate, i.e. free of aliasing shadow map artifacts[1]).

1. Render the view-space linear depth information of the scene into the Eye-Space Depth Buffer, as under Virtual Tiled Shadow Mapping (section 3.3.3 (pg 26)).

2. Use the Eye-Space Depth Buffer bound to a fragment-shader to create

[1] In practical terms that means that for a static light the resulting shadow displays no visible shadow map texels and is stationary even if SM focusing is used, i.e. it behaves as expected from a shadow; another way to put it is, that it looks like a shadow cast by a shadow volume, while still retaining shadow map advantages, such as needing no extra scene information, or being able to shadow non-polygonal or alpha textured objects.

what we call the "**Shadow Map Tile Mapping Map**" ("**SMTMM**"; see Figure 5.3 (pg 55)). The SMTMM contains information for each pixel in the scene about 1) where the pixel will query the shadow map, when inquiring whether it lies in the shadow, and 2) what resolution the shadow map would require along each SM-axis at this position to supply subpixel accuracy when answering the shadow map query.

3. Transfer the SMTMM to CPU memory and process it to create the "**Shadow Map Tile Grid**". The Shadow Map Tile Grid contains information about what resolution each SM-tile of a virtual $n \times n$ Tiled SM would need along each SM-axis to supply subpixel accuracy when used to shadow the scene.
4. Construct the "**Shadow Map Tile Grid Pyramid**" above the Shadow Map Tile Grid, by pulling up the maximum needed SM-tile resolution along each axis.
5. Traverse the Shadow Map Tile Grid Pyramid recursively top down, building an implicit kd-tree of SM-tiles. When the resolution requirement of a such created SM-tile can be satisfied along both SM-axes with a SM-tile-texture with dimensions supported by the GPU, the corresponding SM-tile shadow map is created and immediately used to shadow its part of the scene as under Deferred Shadowing (section 3.3.3 (pg 26)).

The following explains the steps in more detail, and in section 5.12 introduces an important optimization to the basic algorithm:

5.4 Eye-Space Depth Buffer

First, we render the view-space depth information of the scene into the Eye-Space Depth Buffer, as under 3.3.3 (pg 26). For efficiency reasons we again use a $4 \times float$ RGBA-buffer and at the same time render into it the unshadowed RGB color of the scene, so we do not have to rerender the scene (Note: If an application is using depth-first rendering, i.e. starting with a z-only pass, then a $1 \times float$ buffer should be used for the Eye-Space Depth Buffer for the z-only pass, again with a conventional $1/z$-depth buffer attached).

5.5 Shadow Map Tile Mapping Map - Motivation

This section motivates the concept of the SMTMM, and gives a high level description of how its entries are constructed.

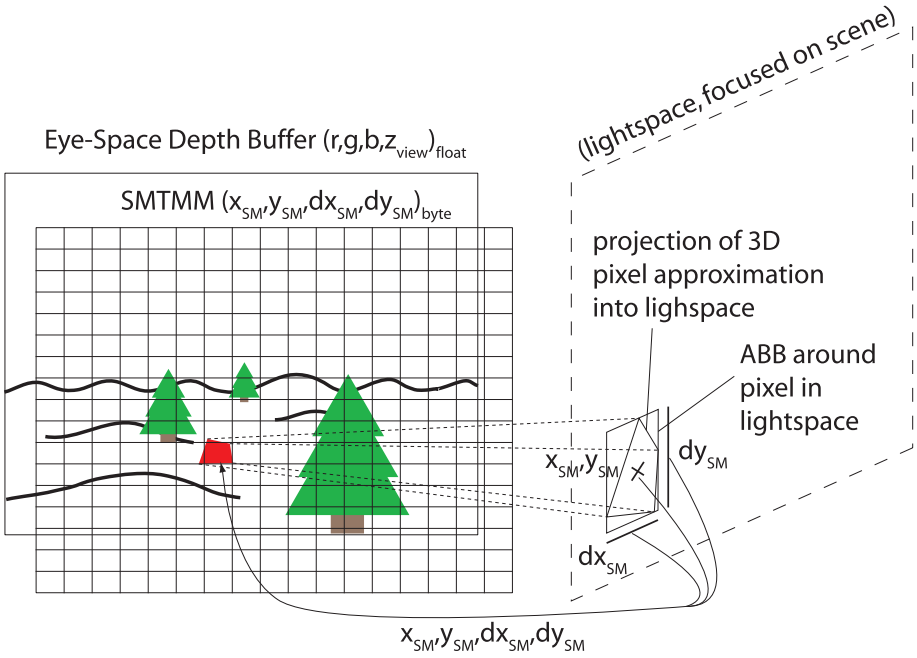


Figure 5.3: Illustration of Shadow Map Tile Mapping Map (SMTMM) creation.

To know beforehand what SM-resolution is needed where in scene we want to calculate for each screen pixel

1. During shadow mapping, at which point will the pixel query (sample) the SM ?
2. What is the extent of the pixel as seen from the light ?

The first question is straightforward to understand: Each pixel corresponds to a 3D-point in world space (where the ray from the eye-point intersects the scene). If we project this point into the SM, we know where the pixel will query the SM.

The second question needs some explaining: What do we mean by "extent of the pixel as seen from the light" ? Each pixel on the screen can be interpreted as a small "**pixel-view frustum**" going from the eye-point through the 4 corners of the pixel into the scene. The pixel actually represents the area in the scene, where the pixel-view frustum intersects the scene (One can visualize this as a spot light located in the eye-point projecting the pixel onto the scene). We are interested in the the size and the orientation of the projection of the pixel into the scene, locally approximated by a planar patch ("**pixel-patch**"[2]).

[2]Note that "patch" is used in its more common meaning here, not in the sense of e.g. a patch in NURBS modeling.

The size of the pixel-patch is related to perspective aliasing, since the less the pixel-view frustum extends into the scene before intersecting it, the smaller the patch that is projected onto a screen pixel will be, the more resolution the SM will need at this spot, so that the SM-texel covering the spot is not larger than the patch.

The orientation of the patch relative to the direction of the light rays is related to projection aliasing, since the more the orientation of the patch is orthogonal to the light direction in the area of the patch, the more resolution the SM will need at this spot, so that the projection of the SM-texel onto the patch does not become larger than the patch.

More mathematically put, what we do is the following: We construct a planar patch approximation of the scene area which is projected onto a screen pixel. Then we project this patch into the SM, getting a quadrilateral in the SM. This quadrilateral needs to have a 1:1 correspondence to a SM-texel in the SM, if the resulting shadow at the scene area of the patch (i.e. at the screen pixel) shall be pixel accurate. To get the required SM-resolution at the point where the pixel will sample the SM from the quadrilateral, we compute an SM-axis-aligned bounding rectangle around it. We store the extent of the quadrilateral bounding rectangle along each SM-axis in the SMTMM, as a measure for the required SM-resolution along each SM-axis at this pixel.

The only remaining question is, how to efficiently construct the pixel-patch ? We do this using the linear depth information (z_{view} stored in the Eye-Space Depth Buffer). First we calculate the positions of the centers of the 4 left/right and upper/lower neighboring pixels of the current screen space pixel, and look up their respective linear depth. Following that, we pick the left or right neighboring pixel which has the larger absolute depth difference to the depth of the current pixel. The reason why we pick the point with the larger absolute difference in depth is that we do not want to miss a surface which is seen at a strafing angle from the eye-point. We then do the same for the upper/lower neighboring pixel, arriving, together with the current pixel center, at 3 points which span a plane which we use to construct the pixel-patch. We do so by calculating the positions of the left/right and upper/lower neighboring pixels on the plane, which we then transform from $(x_{screen}, y_{screen}, z_{view}$ to view space $((x_{eye}, y_{eye}, z_{eye})$ using the Deferred Shadowing unproject transformation 3.1 (pg 28). Applying the inverse view space transform we then arrive at the world space coordinates of the pixel-patch.

Note that due to the neighboring pixels coordinates being calculated in screen space (i.e. post-projective space), the pixel-patch automatically is correctly "perspective un-shortened" through the unproject transformation 3.1, i.e. he is

the larger the farther away he is from the eye-point.

From there we can easily project the pixel-patch into SM-space via the light space transform, arriving at the SM quadrilateral corresponding to the pixel-patch in light space, around which we finally construct the SM-axis aligned bounding rectangle.

Note that due to using the depth information in the vicinity of the current pixel in the calculation of the pixel-patch, the algorithm also works if the SMTMM has a smaller resolution than the screen.

Figure 5.3 (pg 55) shows the SMTMM pixel grid, a corresponding pixel-patch (in red) being projected into the SM, and the resulting information about the SM projected center of the pixel and the pixel-patch bounding rectangle being stored at the SMTMM pixel.

5.6 Shadow Map Tile Mapping Map - Technical Details

The "**Shadow Map Tile Mapping Map**" ("**SMTMM**") is a $4 \times byte$ buffer. One can think of it as being laid on top of the frame buffer, normally having less resolution than the frame buffer, and containing information about the shadow map resolution needs of the scene in the area that each SMTMM "pixel" covers.

The first two byte values in each SMTMM entry ("pixel") contain information about the position where the center of the frame buffer rectangle will query the shadow map; the last two byte entries represent the resolution needed along each SM-axis at the position in the shadow map. We use byte values for the entries to keep the read-back operation and the CPU processing in the next step fast; for the same reason, the SMTMM is normally chosen to have lower resolution than the frame buffer (see the results section for a practical range of values). Using byte values for the shadow map gives us information about the needed SM-resolution discretized to a 256×256 grid of SM-tiles; this is no restriction in practice, since for e.g. 4096^2 SM-tile-textures, a maximum refinement of 256×256 gives subpixel accuracy even for large scenes.

The position in the shadow map is calculated in the pixel shader by transforming the screen-space coordinates (x_w, y_w) of the pixel (passed to the pixel shader as texture coordinates) and the eye-space z (=depth) entry z_e, read from the Eye-Space Depth Buffer, into eye-space (x_e, y_e, z_e) using the matrix given in section 3.3.3, formula 3.1 (pg 28); from there it is transformed into the light space of the shadow map. Since the coordinates will already be in the range

$[0,1]$, simply outputting them to the $4 \times byte$ SMTMM surface will automatically lead to it being converted to byte range ($[0,255]$) by the graphics hardware.

The resolution requirement along each SM-axis is approximated as follows in the pixel shader: First we calculate the (x,y)-coordinates of the neighboring screen space-pixels in x- and y-direction in $[0,1]^2$ (i.e. the left/right and upper/lower screen space neighbors position in texture coordinates) from the texture coordinate of the current pixel passed to the pixel-shader. We then use these texture coordinate to look up the corresponding view-space depth values in the Eye-Space Depth Buffer; from these, we calculate the smaller absolute Δz along the x- and y-axis, Δz_x and Δz_y. We then use these Δz values together with the x,y-coordinates of the neighboring pixels to construct an approximate rectangle representing the current pixel in space. Then we project this rectangle into SM-space, and calculate a SM-axis-aligned bounding box around it. The half length of each of this bounding box's extent, Δ_{sm_axis}, with $sm\_axis = \{0,1\}$, is then used as the base measure for the required SM-resolution along each SM-axis at this point.

To then quantize the needed SM-resolution along the SM-axis into a byte value, we use the following formula:
$-log_2(round(\Delta_{sm\_axis} + float2(0.5,0.5))/256$
(i.e. we output it as a logarithmic value normalized to the range $[0,1]$, which the the graphics hardware again automatically converts to byte range).

The conversion to a logarithmic scale allows us to represent a large scale of required SM-resolutions in a byte value, from 1×1 to $2^{255} \times 2^{255}$.

We add 0.5 before the logarithmic transformation to not get a one-too-small required SM-resolution value when the resulting logarithmic value, normalized to $[0,1]$ by dividing by 256, is quantized to the integer byte range (mapped onto the 255 byte values) when the pixel shader result is written into the SMTMM.

The full SMTMM creation pixel shader can be found in section 5.17 (pg 73).

5.7 Shadow Map Tile Grid Creation

To create the "**Shadow Map Tile Grid**" ("**SMTG**") we then read back the SMTMM to CPU memory. In practice it suffices for the SMTMM to have lower resolution than the frame buffer, e.g. 256×256, which makes the readback and CPU processing fast (note that the equality of the SMTMM dimension of 256 in this example and the number of 256 distinct values in the SMTMM entries is coincidental).

Note that the SMTMM is in screen space whereas the SMTG is in light space.

What we want is a $n \times n$ SM-tile-grid structure, with each grid cell containing the needed resolution along each SM-axis and the **screen space bounding rectangle** for each SM-tile, an axis aligned rectangle around the pixels on screen that are affected by the SM-tile. As in brute-force $n \times n$ Virtual Tiled Shadow Mapping above, n is the maximum number of SM-slices along each SM-axis we would like to allow; a typical value for n would be 16 or 32, corresponding to 256 or 1024 SM tiles for Virtual Tiled Shadow Maps.

The random memory access ability of the CPU is well suited for this task; after having read back the SMTMM, we lock the surface and process each pixel entry: We use the stored information about the SM-tile position to access its corresponding SM-tile-grid cell, and update 1) its needed SM-resolution entries along each SM-axis (minimally by maximizing the existing value with the entries in the SMTMM; see below for details) and 2) its screen space bounding rectangle (through extending it to enclose the pixel position of the current pixel in the SMTMM).

Since the SMTMM generally is chosen to have a much larger resolution than the SM-tile-grid, e.g. 256 entries per axis compared to e.g. 32, data from several SMTMM entries will be accumulated in the same SM-tile-grid cell.

Minimally it would suffice to only record the maximum needed resolution along each SM-axis in each grid cell; however, to be able to allow for discarding very few pixels requiring a large resolution later on (which can come from e.g. a very small area on screen having an orientation which leads to large projection aliasing), we actually count the number of pixels in each grid cell requiring a certain resolution. We use fixed size arrays at each SM-tile-grid-cell to hold the pixel count statistics. To allow us to use fixed size arrays and keep them small, we count the number of pixels below a resolution η_0 and above a threshold resolution η_1 in one array entry respectively, and the number of pixels needing a resolution in between each in their own entry; this is to keep cache locality high, since we are not interested in the detailed statistics of pixels with very small resolution requirements, because evidently they are easy to fulfill, and hypothetical pixels with extremely high resolution requirements, which do not occur in practice. See the results section for practical values for η_0 and η_1.

The following pseudocode illustrates the Shadow Map Tile Grid Creation:

```
// smtg ... instance of the SMTG
// shift ... shift-converts from the SMTMM SM-coordinates
// entries to SMTG ones (e.g. [0,255] => [0,31])
const int shift = 256/smtg.n
// smtmm ... instance of the SMTMM
```

```
// smtmm.n ... extent of SMTMM along both axes
for ix_smtmm = 0 to smtmm.n - 1
 for iy_smtmm = 0 to smtmm.n - 1
  SMTMM_Cell c_smtmm = smtmm(ix_smtmm,iy_smtmm)
  SMTG_Cell c_smtg =
   smtg(smtmm.ix_sm >> shift,smtmm.iy_sm >> shift)
  // Update the screen-space, axis aligned bounding box
  // around the SM-tile
  c_smtg.abb_screen.ExpandToIncludePoint(
   ix_smtmm/smtmm.n,iy_smtmm/smtmm.n
  )
  // Update the maximum needed SM-resolution
  c_smtg.sm_res_x = MAX(c_smtg.sm_res_x,c_smtmm.sm_res_x)
  c_smtg.sm_res_y = MAX(c_smtg.sm_res_y,c_smtmm.sm_res_y)
```

5.8 Shadow Map Tile Grid Pyramid Creation

After we have filled the SM-tile-grid with data from the SMTMM, we proceed by building a pyramid ("**Shadow Map Tile Grid Pyramid**", "**SMTGP**") of SM-tile grids on top of it, where each successive grid has halved dimensions of its predecessor and the needed resolution along each SM-axis is the maximum of the corresponding 2×2 grid cells in the predecessor grid; i.e. we pull up the needed resolution along each SM-axis by replacing 2×2 cells with one cell in the next smaller grid, containing the maximum value of each of the 4 cells and the screen space bounding rectangle around all 4 bounding rectangles.

Note that the "needed resolution along each SM-axis" refers to a theoretical single shadow map focused on the scene, with a resolution large enough to give subpixel accurate shadows; replacing the $2 \times 2 = 4$ values with their maximum in the parent cell in the next higher Shadow Map Tile Grid Pyramid level is therefore not a heuristic, but a mathematically exact operation (The 1×1 top level grid then contains the resolution needs for this theoretical single shadow map; as can be seen in the Results section, it grows to dimensions ≥ 131072 ($= 16 \times 8192$) even for medium-sized scenes, which would require ≥ 64 GB of GPU memory).

The Shadow Map Tile Grid Pyramid Creation in pseudocode:

```
// smtg ... instance of the initial SMTG
// smtg.n ... extent of SMTG along both axes
// i_pyramid ... SMTGP index
const int i_pyramid = log2(smtg.n)
```

```
while(i_pyramid > 0)
 // smtgp ... instance of the SMTGP
 smtgp(i_pyramid) = smtg
 for ix = 0 to smtgp(i_pyramid).n - 1
  for iy = 0 to smtgp(i_pyramid).n - 1
   SMTGP_Grid_Cell c_curr = smtgp(i)(ix,iy)
   SMTGP_Grid_Cell c_parent = smtgp(i-1)(ix >> 1,iy >> 1)
   // Update the screen-space, axis aligned bounding box
   // around the parent SM-tile
   c_parent.abb_screen.ExpandToIncludeABB(c_curr.abb_screen)
   // Update the maximum needed SM-resolution
   c_parent.sm_res_x = MAX(c_parent.sm_res_x,c_curr.sm_res_x)
   c_parent.sm_res_y = MAX(c_parent.sm_res_y,c_curr.sm_res_y)
   i_pyramid = i_pyramid >> 1
```

5.9 Shadow Map Tile Grid Pyramid Traversal

Finally we traverse the grid pyramid top down, building an implicit kd-tree as we recursively traverse it, as follows: If the resolution requirement of the SM-tile-grid cell along at least one axis cannot be satisfied with a SM-tile-texture with dimensions supported by the GPU (e.g. on current GPUs typically: required SM-dimension $>$ 4096), we split it symmetrically along one or both SM-axis into 2 or 4 subcells. We split into 2 subcells if only one axis has SM resolution requirements which cannot be fulfilled, otherwise we split into 4 subcells. Otherwise we use Deferred Shadowing (see Section 3.3.3) immediately creating the SM-tile with the required resolution along each axis and shadowing the Shadow Result Texture (see next paragraph) with it, using the Eye-Space Depth Buffer to get the depth values of the scene, as described in section 3.3.3.

The "**Shadow Result Texture**" again (see Section 3.4 (pg 29)) is a $1 \times byte$ texture with the same dimensions as the frame buffer, into which we write only the results of the shadowing operation.

Pseudocode for the traversal of the Shadow Map Tile Grid Pyramid:

```
// SMT ... SM-tile instance
// P ... SMTGP pos index + pyramid index
// smtq ... queue holding SMT
smtq.push(SMT(P(0,0),P(0,0)))
while(!smtq.empty())
 SMT smt = smtq.pop()
 int ip_x = smt.ip_x, int ip_y = smt.ip_y
 int sx = max(0,ip_y-ip_x), int sy = max(0,ip_x-ip_y)
```

```
Rect rect(ix << sx,iy << sy, ((ix+1) << sx)-1,((iy+1) << sy)-1)
// ex and ey are 0 for no further refinement, 1 otherwise
int ex = RefineQ(smtgp(
  MAX(ip_x,ip_y)).MaxSmResInRect(rect).sm_res_x,ip_x,framebuffer.nx)
int ey = RefineQ(smtgp(
  MAX(ip_x,ip_y)).MaxSmResInRect(rect).sm_res_y,ip_y,framebuffer.ny)
if(ex > 0 || ey > 0) // refine this SM-tile further
 int ip_x_sub = smt.ix + ex, int ip_y_sub = smt.iy + ey;
 int ix_sub = smt.ix << ex, int iy_sub = smt.iy << ey;
 for diy=0 to ey
  for dix=0 to ex
   smtq.push(SMT(P(ix_sub+dix,ip_x_sub),P(iy_sub+diy,ip_y_sub)))
else  // do not refine this SM-tile further
 ShadowShadowResultTextureWithSmTile(smt)
```

with

```
// sm ... SM-texture
RefineQ(sm_res_needed,i_refinement,framebuffer_nx_or_ny) {
 return sm_res_needed > i_refinement -
  log2(sm.n) - round(log2(framebuffer_nx_or_ny/smtmm.n)+0.5)
}
```

5.10 Apply Shadow to Scene

In a final step we combine[3] the scene RGB from the Eye-Space Depth Buffer with the Shadow Result Texture, and write the resulting shadowed scene into the frame buffer.

The whole algorithm not only leads to a greatly reduced number of SM-tiles that need to be created compared to the brute-force approach, but also uses smaller and rectangular SM-tile textures for farther away SM-tiles. It therefore makes much higher quality shadow maps possible in real-time. Please see the Results section for a quantitative comparison.

5.11 Quality vs Performance Parameter

Fitted Virtual Shadow Maps allow for the introduction of a very intuitive quality vs performance parameter ξ: Subtracting an integer number ξ from the logarithmic resolution requirement value coming from the SMTMM, when deciding

[3] e.g. modulate the scene RGB with the Shadow Result Texture

whether to further refine a SM-tile, allows us to intuitively influence the quality of the resulting shadow in the scene; the larger ξ, the fewer tiles will be created and the better the performance will be. This allows the algorithm to be tuned to a wide range of hardware. Note that the influence of the parameter is smooth, in the sense that it influences the shadow quality of the whole scene in the same way. If ξ is chosen to be large enough, so that only one SM-tile is created, then the shadow quality of Fitted Virtual Shadow Mapping is the same as normal shadow mapping.

5.12 Shadow Map Tile Texture Size Optimization

The basic FVSM algorithm refines the shadow map until the required resolution of each SM-tile along each SM-axis is small enough that it can be satisfied with the maximum quadratic SM texture size which the GPU can handle. In practice, the resolution needs along the SM-tile axes often actually do *not* require a SM-texture which is quadratic and has the maximum GPU supported resolution, but instead a rectangular SM-texture with less resolution could be used. This can be understood from the fact, that there will always be perspective shortening in the scene, i.e. the pixels farther away from the eye-point will always require less SM-resolution (Note that due to the splitting of the SM into tiles this is always the case for farther away tiles, independent of the view direction relative to the light direction, contrary to SM reparametrization techniques, which can only profit from the perspective shortening for light directions which are not parallel or antiparallel to the view direction); projection aliasing can of course counteract this, but even then, the projection aliasing does not normally influence both SM-axes at the same time.

This leads to the optimization that, to reduce the fillrate requirements of the algorithm, instead of using a quadratic maximum-sized texture for all SM-tiles, we create a rectangular SM according to the resolution requirements along each axis. There are 3 different ways to do this:

1. Render into a sub-rectangle of the same quadratic, maximum sized texture.
2. Render into a sub-rectangle of a series of quadratic power-of-two shadow map textures, where each texture in the series has halved dimensions relative to its predecessor.
3. Render into shadow map textures with the exact needed dimensions (= resolution) along each SM-axis.

At first glance it might seem obvious that the first approach is the best, since it evidently requires the minimum amount of GPU memory. However, it is not the fastest approach, which can be seen by realizing that the rows of a texture are usually arranged consecutively in GPU memory; this means, that memory and cache coherence suffer, when one renders to a sub-rectangle of a texture which has a larger width than the sub-rectangle (see Figure 5.5 (pg 66) to see how much performance difference there is in practice).

Due to this, choosing the right approach becomes a trade-off between speed and GPU memory consumption. The following analyzes the memory requirements of the different approaches.

The memory requirements of the 4 different approaches are as follows (with t_{max} being the maximum texture dimension, and w and h being the needed minimum width and height SM texture dimensions to satisfy the SM-tile resolution requirements):

Optimization	SM dimension	mem usage	for $t_{max} = 4096$
none	${t_{max}}^2$	1	64 MB
#1	$w \times h$ sub-rect of ${t_{max}}^2$	1	64 MB
#2	$w \times h$ sub-rect of $max(w,h)^2$	$4/3$	85 MB
#3	$w \times h$	$2 \cdot 4/3$	171 MB

This can be seen, by observing that the for optimization 2, the required SM textures have dimensions: $\{{t_{max}}^2, (\frac{1}{2}t_{max})^2, (\frac{1}{4}t_{max})^2, \ldots\}$ leading to the series (with ${t_{max}}^2$ pulled out) $\sum 1 + \frac{1}{4} + \frac{1}{16} + \ldots = \sum_{n=0} (\frac{1}{4})^n \leq \frac{1}{1-\frac{1}{4}} = \frac{4}{3}$.

For optimization 3, we arrive at the series (${t_{max}}^2$ pulled out again)
$\sum (1 \cdot 1 + \frac{1}{2} \cdot 1 + 1 \cdot \frac{1}{2}) + (\frac{1}{2} \cdot \frac{1}{2} + (\frac{1}{2} \cdot \frac{1}{2}) \cdot \frac{1}{2} + \frac{1}{2} \cdot (\frac{1}{2} \cdot \frac{1}{2})) + \ldots =$
$2 \cdot \sum 1 + \frac{1}{4} + \frac{1}{16} + \ldots = 2 \cdot \frac{4}{3}$.

The effective memory consumption given for $t_{max} = 4096$ is for $1 \times float$ SM textures.

Evidently what we want is a combination of small GPU memory consumption together with good performance; please see the Results section below for a performance comparison of the different approaches and a resulting recommendation which optimization variation to use in practice.

5.13 Handling Semitransparent Objects

When using deferred shadowing semitransparent objects must be treated separately, due to the fact that only the depth entry of the foremost opaque pixel is

stored in the Eye-Space Depth Buffer.

The following lists several approaches to handle semitransparent objects in a scene together with FVSMing:

1. A trivial solution is evidently to render the semitransparent objects after having shadowed the scene without shadowing them; this will give them a (slight) glow effect, which might or might not be acceptable.
2. A second approach would be to again render the semitransparent objects after having shadowed the scene, shadowing them with with conventional shadow mapping (i.e. using a single shadow map). In practice, semitransparent objects are usually implemented to only receive shadows, but not cast them, due to the increased complexity of the shadowing problem when semitransparent objects are involved. The fact that in this case no self shadowing can occur makes this approach practical. Also note that the shadow (and therefore any artifacts) will be less visible the more transparent the object is.
3. Less practical but more accurate solutions would work along the line of rendering the semitransparent objects to a separate buffer storing the color and linear depth of the foremost semitransparent object. One could then use this buffer later on to also shadow the foremost semitransparent object correctly. Note that in this case one also needs twice the number of entries int the SMTMM.
4. Finally, a completely different approach would be to not use Deferred Shadowing, but Multi-Pass Shadowing (see Section 3.3.2 (pg 25)) to apply the SM-tiles to the scene, sped up by restricting the part of the scene that needs to be rerendered to the screen space extent (see first paragraph of section 5.8 above) of each SM-tile. In this case the semitransparent objects can be rendered and shadowed (with each SM-tile) as if only a single shadow map were being used.

5.14 Results

Unless otherwise noted, all results were created on an NVidia GeForce 8800GTS with 640 MB of RAM and a Pentium4 3.4 GHz (2 GB RAM).

The aim of our work is to research the applicability and improve the quality of dynamic shadow map shadows when applied to large to medium-sized scenes. In practice, the resolution of shadow maps currently supported in hardware suffices to create shadows for light sources with a small area of influence (spotlight

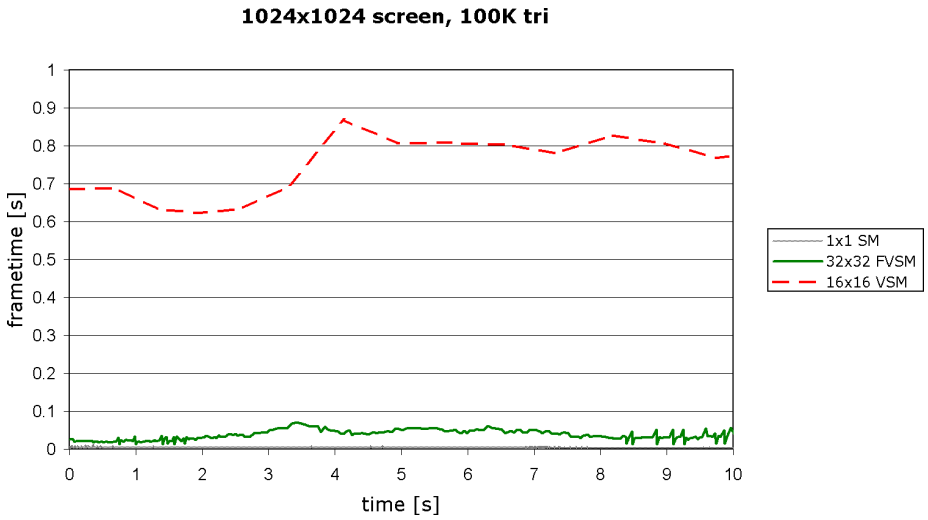


Figure 5.4: Performance comparison between normal, Virtual Tiled and Fitted Virtual SMing along path in forest test scene. The FVSM curve was done using SM-tile texture size optimization #2 (see Section 5.12 (pg 63)).

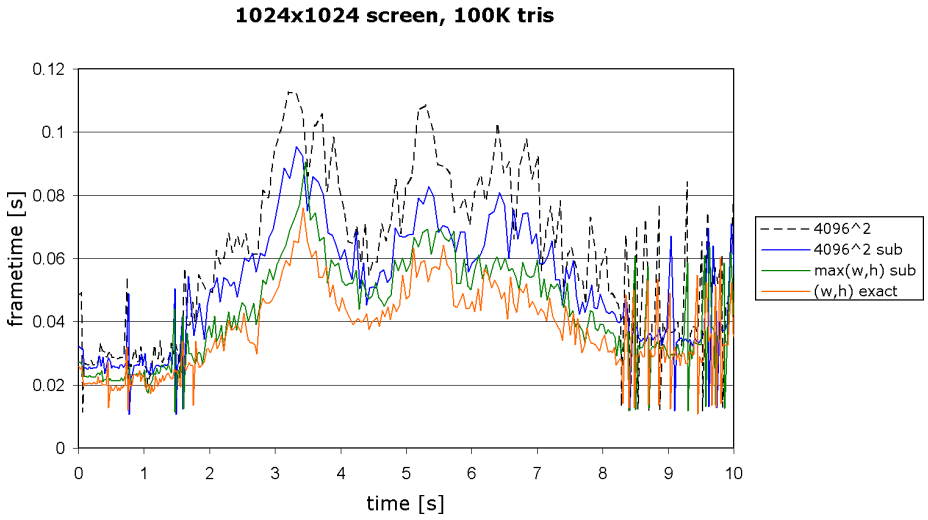


Figure 5.5: FVSM performance comparison between different SM-tile creation modes (see Section 5.12 (pg 63)). One can see that each optimization smoothes the frame time peaks by a larger amount, getting closer to the general real-time rendering goal of a constant framerate.

located near the ground with a small spread angle, point light lighting a small room, etc), especially when combined with shadow map reparametrization. In addition, evidently the render costs of an algorithm such as FVSMs, although being much faster than brute-force Virtual Tiled SMing, are currently too high to apply it to several light sources (see Figure 5.4 for a frametime ccomparsion).

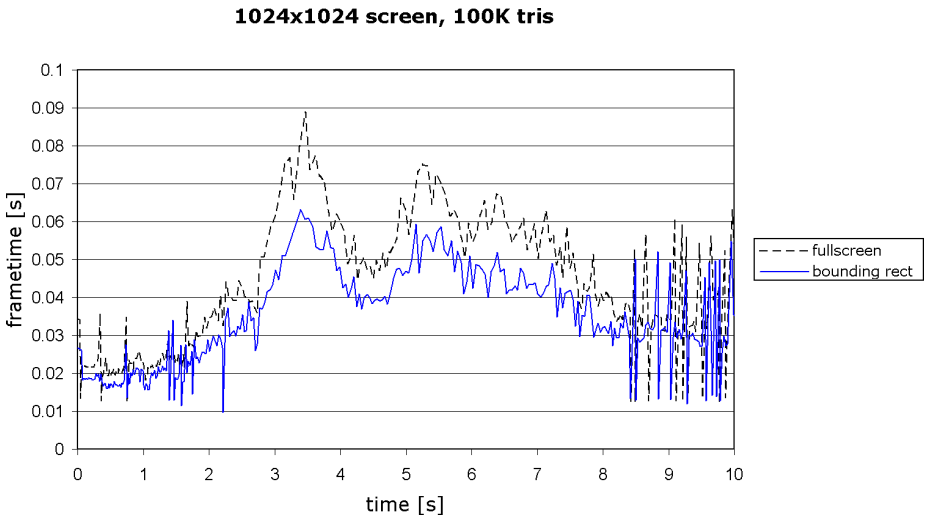


Figure 5.6: FVSM SM-tile application to Shadow Result Texture: Comparison between using a fullscreen rectangle and using the SM-tile screen space bounding rectangle. Again one can see that the optimization has the beneficial effect of smoothing the frametime peaks.

Our test scene of broad-leaf trees on a hilly terrain was chosen accordingly: It is a medium-sized outdoor scene, which makes it of practical interest, since dynamically shadowing outdoor scenes in high quality continues to be a challenge. It will in many cases be lit by the sun, i.e. a single light source which can be modeled by a directional light or far away spotlight, and which can therefore be shadowed using a single shadow map (preferably of a very high resolution). It also lends itself to be efficiently modeled and rendered using alpha textured geometry, and can therefore not be shadowed e.g. by shadow volumes; in our case the fence and the leaves on the trees are alpha textured quads (this could be extended to e.g. using billboards or billboard clouds for farther away trees). Additionally, animating the direction of the sunlight turns it into a fully dynamic scene from the point of view of shadowing it (Note that even if the scene itself is completely static, any change in the light position, direction etc would result in invalidating any cached shadow map information, making it unsuited for algorithms that are built on caching shadow map information from previous frames to work efficiently).

Figure 5.1 shows a screenshot from our test scene; it compares the quality of normal shadow mapping (4096^2 shadow map) with Fitted Virtual Shadow Maps (lightspace perspective shadow maps (LiSPSM) active in both cases).

Figure 5.4 shows frame time curves from a path through a forest scene with 10^5 triangles rendered into a 1024×1024 frame buffer

(you can download the videos of the flythrough along the path at http://www.cg.tuwien.ac.at/research/vr/fvsm). The uppermost curve depicts brute-force Virtual Tiled SMing, using 16×16 4096^2 shadow maps; the FVSM curve shows Fitted Virtual SMs with adaptive SM-tile shadow map textures size optimization #2 (see Section 5.12 (pg 63)) and 32×32 maximum refinement. The 1×1 SM-curve finally gives the frame times for conventional SMing using a 4096^2 shadow map texture (leading to greatly reduced shadow quality). LiSPSM was active for all renderings. Figure 5.2 shows screenshots at the end of the path through the forest scene

Figure 5.5 shows a performance comparison (along the same path in the scene) between different FVSM SM-tile creation modes (see Section 5.12). The important thing one can see is, that this optimization smoothes the frame-time peaks, i.e. it improves the worst case performance of the algorithm. One can also see that each of the SM-tile creation modes is faster than its predecessor. Since the memory consumption of the 3^{rd} scheme relative to the first two is $4/3$ (i.e. it consumes only one third more memory), in practice rendering to a sub-rectangle of a series of quadratic power-of-two shadow map textures is the best compromise between performance and memory consumption in most cases.

Figure 5.6 compares the performance of applying the SM-tiles to the Shadow Result Texture using a fullscreen quad or the screen space bounding rectangle around each respective SM-tile (see Section 5.8). Note that again the frame-time peak at around $3.5s$ is smoothed, improving the worst case performance of the algorithm by around 25%.

For a 1024^2 frame buffer we found that a 256^2 SMTMM gave good results, while at the same time keeping the SMTMM creation and processing overhead low.

For the detailed SM-tile-required-resolution-statistics parameters η_0 and η_1, we have found that $\eta_0 = 17$ and $\eta_1 = 22$ work well in practice.

Section 5.16 (pg 70) below discusses the influence of the quality vs performance parameter ξ for $\xi = 0, \ldots, 5$, where $\xi = 5$ gives equivalent shadow quality to normal shadow mapping. One can see that the shadow quality decreases homogenously in the whole scene.

The absence or presence of undersampling artifacts when using shadow mapping with focusing can be best seen in motion, so please see some videos and screenshots on the corresponding webpage[4].

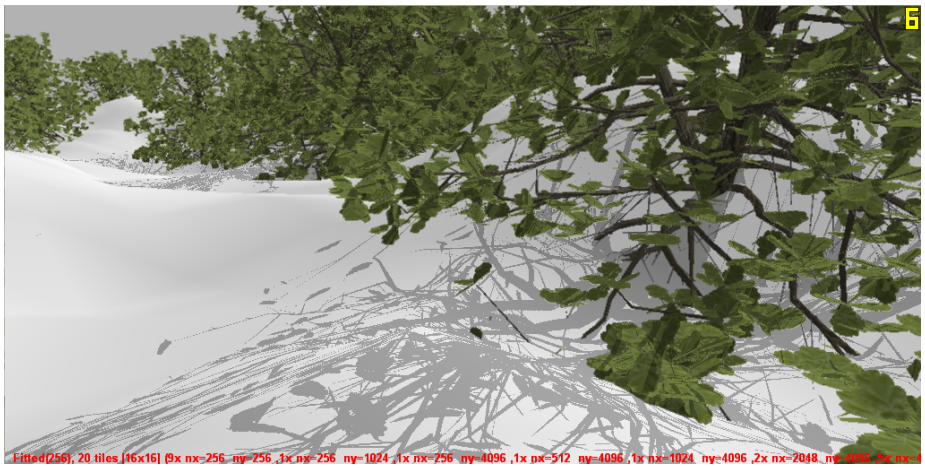
[4] http://www.cg.tuwien.ac.at/research/vr/fvsm

5.15 Conclusion

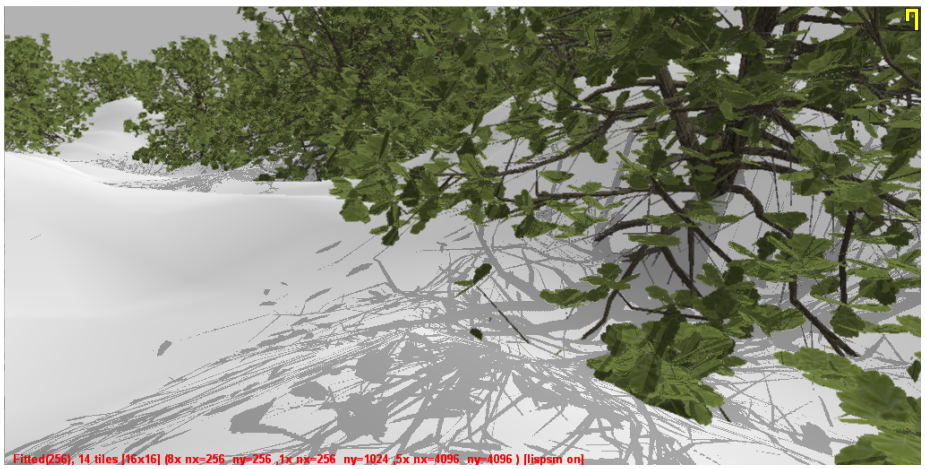
This chapter described Fitted Virtual Shadow Maps, a new smart shadow map algorithm which allows for the efficient shadowing of large scenes without undersampling artifacts, while at the same time making use of previous shadow map improvements, such as shadow map focusing and shadow map reparametrization techniques. Virtual Shadow Mapping allows the algorithm to bypass the memory cost and texture size limits of current GPUs. Instead of brute-force Virtual Tiled Shadow Mapping, Fitted Virtual Shadow Maps employ a combination of GPU and CPU processing to create a map which contains information about what resolution would be required where in a shadow map to give subpixel accuracy when shadowing the scene. This leads to a performance increase of at least an order of magnitude over the brute-force approach, while still greatly reducing or even removing perspective *and* projection aliasing. The algorithm can be tuned with several parameters according to the quality requirements of the scene; most importantly the shadow quality can be uniformly reduced down to normal shadow mapping by use of a very intuitive quality-vs-speed parameter.

5.16 Influence of ξ

The following screenshots show the influence of the FVSM quality vs performance parameter ξ on the graphical quality of the resulting shadow and performance. One can see that with increasing ξ, the shadow quality is reduced homogenously in the whole scene.

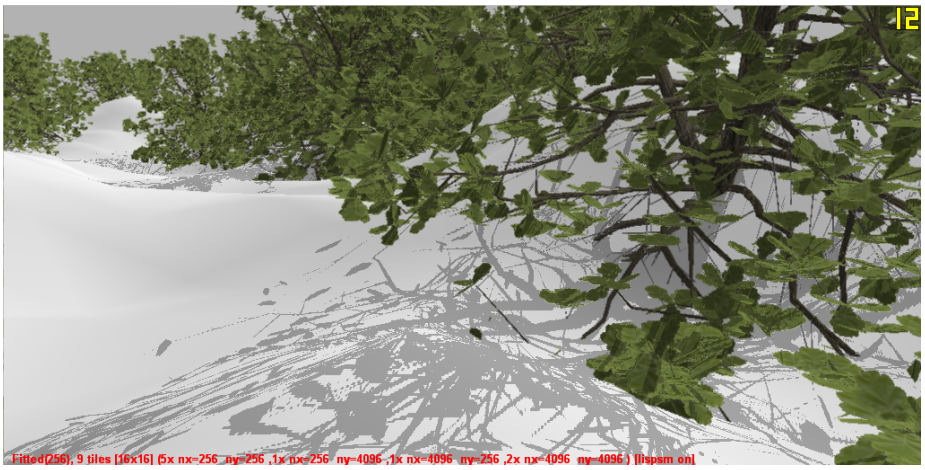


(a) FVSM $\xi = 0$

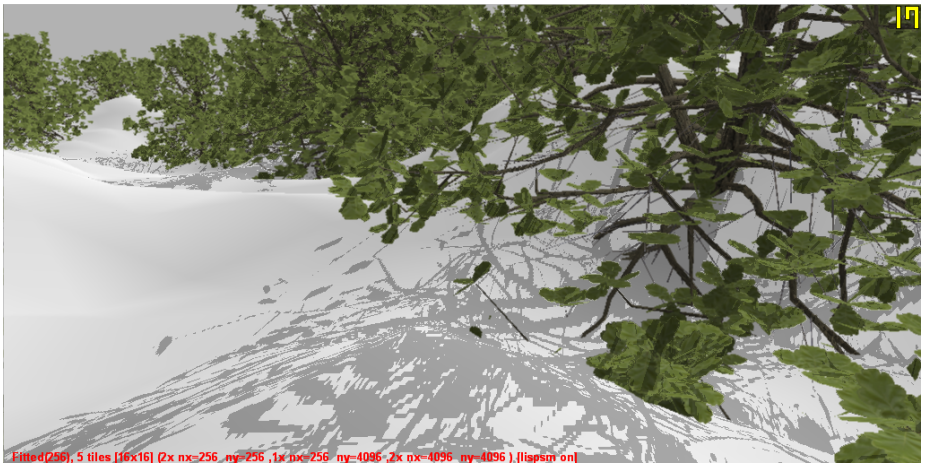


(b) FVSM $\xi = 1$

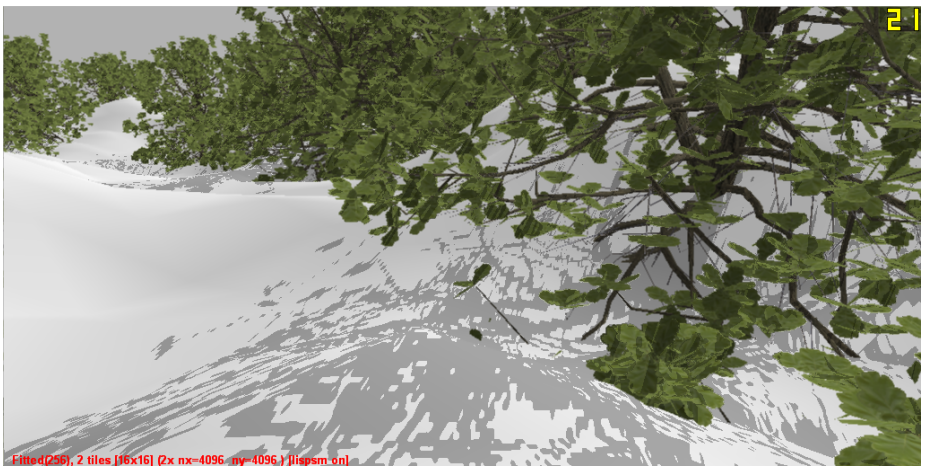
Figure 5.7: Influence of FVSM quality vs performance parameter ξ.



(c) FVSM $\xi = 2$

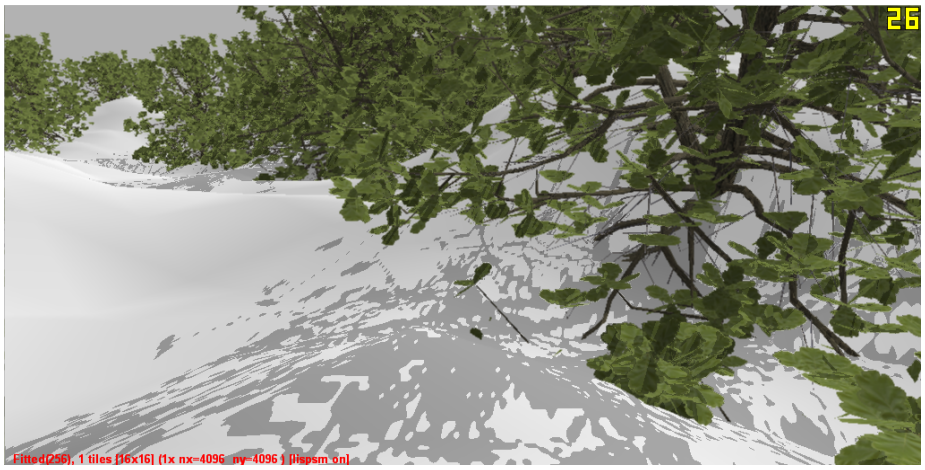


(d) FVSM $\xi = 3$

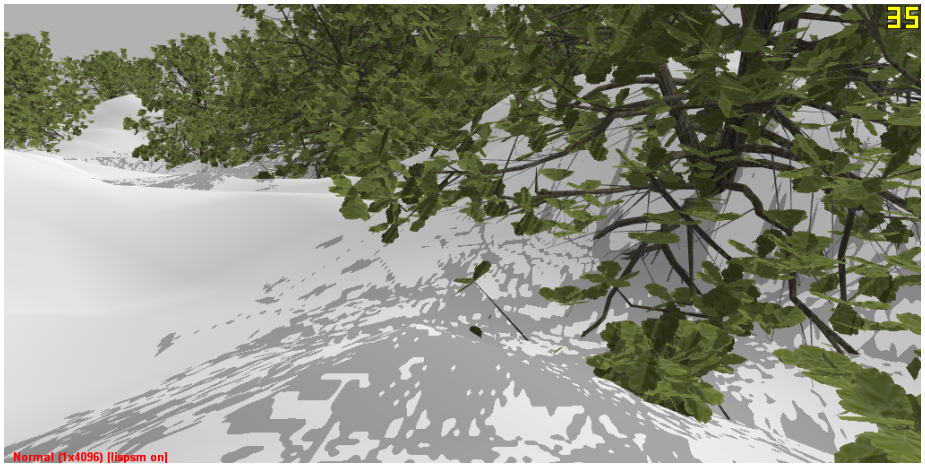


(e) FVSM $\xi = 4$

Figure 5.7: (continued) Influence of FVSM quality vs performance parameter ξ.



(f) FVSM $\xi = 5$



(g) normal SMing

Figure 5.7: (continued) Influence of FVSM quality vs performance parameter ξ. The last screenshot shows 4096^2 normal shadow mapping for comparison; one can see that FVSM with $\xi = 5$ above gives the same quality.

5.17 SMTMM Creation Pixel Shader

```
Ps_OUT PsCreateShadowMapTileMapping(Ps_IN IN)
{
    Ps_OUT OUT;

    // texture coordinate of center of current pixel
    float2 tc_pixel_center = IN.v2_tc.xy;

    // texture coordinates of right,left,upper and lower neighbor
    // of current pixel, clamped to rendertarget extent
    float2 tc_pixel_neighbor_right =
        float2(clamp( tc_pixel_center.x +
        rendertarget_nr_pixel_inv.x, 0, 1), tc_pixel_center.y );
    float2 tc_pixel_neighbor_left =
        float2(clamp( tc_pixel_center.x -
        rendertarget_nr_pixel_inv.x, 0, 1), tc_pixel_center.y );
    float2 tc_pixel_neighbor_upper =
        float2(tc_pixel_center.x, clamp( tc_pixel_center.y +
        rendertarget_nr_pixel_inv.y, 0, 1));
    float2 tc_pixel_neighbor_lower =
        float2(tc_pixel_center.x, clamp( tc_pixel_center.y -
        rendertarget_nr_pixel_inv.y, 0, 1));

    // read viewspace z for current pixel from \ESDB{}
    float z_view_center = tex2D(tex_shadow_depth_buffer, tc_pixel_center).w;

    float2 v2_dz_view_use = float2(
            sm_tile_mapping_pick_smaller_dz(z_view_center,
                tc_pixel_neighbor_left,tc_pixel_neighbor_right),
            sm_tile_mapping_pick_smaller_dz(z_view_center,
                tc_pixel_neighbor_lower,tc_pixel_neighbor_upper)
        );

    // pos of left neighbor of current pixel in the shadowmap
    // ScreenspaceToShadowmapCoordinatesAndLightspaceDepth uses
    // the "Deferred Shadowing" matrix to transform to eye-space.
    float3 pos_sm_left = ScreenspaceToShadowmapCoordinatesAndLightspaceDepth(
            tc_pixel_neighbor_left.x, tc_pixel_center.y,
            z_view_center - v2_dz_view_use.x
        );

    // pos of right neighbor of current pixel in the shadowmap
    float3 pos_sm_right = ScreenspaceToShadowmapCoordinatesAndLightspaceDepth(
            tc_pixel_neighbor_right.x, tc_pixel_center.y,
            z_view_center + v2_dz_view_use.x
        );

    // pos of lower neighbor of current pixel in the shadowmap
    float3 pos_sm_lower = ScreenspaceToShadowmapCoordinatesAndLightspaceDepth(
```

```
            tc_pixel_center.x, tc_pixel_neighbor_lower.y,
            z_view_center - v2_dz_view_use.y
        );

    // pos of upper neighbor of current pixel in the shadowmap
    float3 pos_sm_upper = ScreenspaceToShadowmapCoordinatesAndLightspaceDepth(
            tc_pixel_center.x, tc_pixel_neighbor_upper.y,
            z_view_center + v2_dz_view_use.y
        );

    float2 pos_sm_max =
        Max(pos_sm_left.xy,pos_sm_right.xy,pos_sm_lower.xy,pos_sm_upper.xy);
    float2 pos_sm_min =
        Min(pos_sm_left.xy,pos_sm_right.xy,pos_sm_lower.xy,pos_sm_upper.xy);

    // Approximate extent of the current pixel projected onto the shadowmap
    float2 dxy_pixel_on_shadowmap = 0.5 * (pos_sm_max - pos_sm_min);

    // Measure of resolution needed to shadow this pixel with subpixel accuracy
    float2 pixel_shadowmap_resolution_measure;

    // use pixel_shadowmap_resolution_measure =
    // -log2(round(dxy_pixel_on_shadowmap + float2(0.5,0.5))/256
    frexp( sqrt(2.0) * dxy_pixel_on_shadowmap,
        pixel_shadowmap_resolution_measure);

    //pixel_shadowmap_resolution_measure *= (1.0/256.0);
    // [0,255] => [0,1] (for output to 8-bit surface)
    pixel_shadowmap_resolution_measure =
        ldexp(-pixel_shadowmap_resolution_measure, -8);

    // Postion of pixel center in the shadowmap
    float3 pos_shadowmap =
        ScreenspaceToShadowmapCoordinatesAndLightspaceDepth(
            tc_pixel_center.x, tc_pixel_center.y, z_view_center
        );

    // Output SM-tile position and resolution measure along SM x- and y-direction.
    OUT.color =
        float4(pos_shadowmap.x, pos_shadowmap.y,
            pixel_shadowmap_resolution_measure.x,
            pixel_shadowmap_resolution_measure.y
        );

    return OUT;
}
```

Chapter 6

Discrete LODs

6.1 Introduction

The previous chapters have dealt with increasing the visual quality of shadows produced by shadow mapping, by increasing the resolution supplied by the shadow map used to shadow a scene in a smart manner. The following two chapters deal with another problem of visual quality, this time in the temporal domain: discrete LOD switching.

Discrete level-of-detail (LOD) rendering is a well known acceleration technique where complex objects are replaced by successively simpler (in geometry and/or shading) representations the farther they are from the eye point. During runtime, the renderer chooses from a series of so-called *levels of detail* (LODs) for each object.[1].

While discrete LOD creation and selection have been widely researched, *LOD switching*, i.e., the question how to stage the transition between two different discrete levels of detail of an object, is a significant open problem in using discrete LODs, as evidenced by popping artifacts due to “hard” (i.e. instantaneous) LOD switching in state-of-the-art computer games such as Far Cry, Half-Life 2 or F.E.A.R..

Since LOD switching seems to be a simple task, we have found that most computer graphic researchers assume that the straightforward approach of doing a conventional $(\alpha, 1\text{-}\alpha)$-blend between different levels of discrete LODs solves the problem. However, this approach does not work, as can be

[1] This *LOD selection* is often based either on the distance of the object from the observer, or on an estimate of the number of projected pixels of the object [FS93]. A detailed and comprehensive discussion of most level-of-detail techniques can be found in [LRC+02]

seen by considering the intermediate states of such a blend, where both LOD representations—and therefore the whole object—are rendered semi-transparently, therefore blending with previously rendered objects and/or the background-color.

The blend could be done using offscreen buffers, but this would have a high performance and memory usage penalty. Another commonly held opinion is that the problem can be solved by simply hard-switching at a large enough distance; while this may be true in a mathematical sense for purely geometric LODs that do not depend on the environment (e.g., via shaders), we argue that in practice this does not work due to the nature of the human perception system and that it defeats the purpose of LODs.

6.2 LOD Switching

Several solutions have been proposed for the problem of *LOD switching*, i.e., the question how to stage the transition between two different discrete levels of detail of an object[2]:

6.2.1 Hard Switching

The simplest approach is *hard switching*, i.e., one LOD is replaced by another at some point in time. Although simple and fast, due to the nature of human perception this method leads to very noticeable and visually disturbing temporal discontinuity artifacts (“popping”).

6.2.2 Late Switching

One seemingly obvious solution is so-called *late switching*, i.e., to hard-switch at a “large enough” distance, where the difference between the two LODs is “no longer noticeable”.

There are several problems with this approach, which make it unfit for practical use:

It tries to fulfill two conflicting goals: on the one hand, increasing the frame rate to keep the application real time by switching as early as possible, and on the other hand, trying to reduce popping artifacts by switching as late as possible.

[2] more precisely: between its LOD representations; we are using LODs and their representations interchangeably for brevity reasons.

To *switch as early as possible* is necessary because the number of objects (and therefore the render cost) in a typical 2.5D scene (with a homogeneous object distribution) for which a more complex LOD has to be rendered increases quadratically with the switching distance. This is because the number of objects with distance smaller than r to the viewpoint is $\propto r^2$.

In addition, fill rate for costly shaders used for nearer LODs of an object is only saved when the object still covers a significant amount of screen space when the switch occurs.

As if this were not enough by itself, the second, contradicting goal to *reduce popping by switching as late as possible*, can in general not even be attained in itself, since the human visual system is fine-tuned to notice even small discontinuous changes. This is problematic because two LODs, even when rendered far from the viewpoint, cannot be guaranteed to produce identical images. The goal becomes completely unattainable when different shading effects (e.g., environment maps vs. simpler shading, or rendering specular highlights vs. diffuse shading only) are used for the different LODs, which is frequently the case in today's games.

We conclude that *late switching* is not a practical approach. On the contrary, one wants to switch to lower LODs as *early* as possible in order to make good use of LODs.

6.2.3 LOD Blending in Image Space

LOD *blending in image space* is often mentioned as a generic way to do the transition between LODs. However, just linearly blending the two LODs is not possible, since an incorrect, semitransparent object would result during the blend, when both LODs get rendered semitransparently into the scene (see figure 7.1 and the 2^{nd} paragraph in Section 7.2).

A more complex variation of this method would be to render the two LODs opaquely into separate offscreen buffers and do an $(\alpha, 1\text{-}\alpha)$-blend between them: The render cost to do so is high because it requires two offscreen buffers (+ depth buffers), the size of the frame buffer, combined with costly render target switches to the offscreen buffers and back to the frame buffer to render the result of the LOD blend into the scene. This also requires depth buffer writes in the pixel shader, prohibiting modern hardware depth buffer speedup techniques, such as hierarchical depth buffers.

Multisampling hardware (as was supported by the SGI Infinite Reality [EJ00]), produces worse results than our approach, since the number of stages in the

transition is limited to the number of samples in the multisample mask.

6.2.4 Geomorphing

Some mesh representations allow a technique called *geomorphing* [Hop96, Hop98], where vertex positions are interpolated between the two LODs. This requires that each vertex of one LOD can be uniquely identified with a vertex of the second LOD, which is only possible for special multiresolution mesh representation techniques. While geomorphing is widely referred to as the highest-quality LOD-switching technique, it can make solid structures (like terrains) appear to be moving in a quasi-organic way, and this effect can be as disturbing as hard switching itself (although recent approaches have brought significant improvements in this area, for example for terrain visualization applications [CGG+03]). While a lot of very interesting research has been done in this field, due to the comparatively large implementation effort required, and problems to get it to work fast with modern graphics hardware, very few computer games use geomorphing. Note that even if the geometry transition can be made fairly smooth, appearance switches (e.g., less complex shaders, which might be even more important than geometric simplification in today's games) are still hard switches and will therefore pop. Also, progressive representations do not allow completely different LOD representations (e.g., disjointed vs. skinned LODs). Discrete LODs do not suffer from these limitations, and can stage such transitions smoothly using the method presented in the following chapter.

6.3 Popping Examples in Current Games

This section discusses screenshots taken from current, successful computer games, which all use discrete LODs with hard switching, and therefore exhibit popping artifacts.

Popping is a phenomenon linked to our temporal perception, it is very obvious when perceived in action, but can be hard to spot when just comparing two screenshots on a page; therefore please check the Unpopping webpage[3] to perceive the hard switching dynamically[4,5]

[3] http://www.cg.tuwien.ac.at/research/vr/unpopping/examples

[4] The images on the page take a few seconds to load. Move your mouse over/out of the images to observe the popping.

[5] If you view this thesis electronically, you will also find direct links to the online images under the popping screenshots found on the following pages.

Figure 6.1 shows 2 screenshots taken in computer game Far Cry[6] (released in March 2004): One can see that the representation of the small palm tree in the center of the screen changes instantly from a decal imposter (i.e. a textured quad) to a 3D mesh. The Unpopping algorithm presented in the next chapter can handle smooth switching between different LOD representations, such as decal imposter to 3D mesh. Far Cry uses discrete LOD for all its environment and objects, therefore popping artifacts occur all the time when the player moves.

Screenshots from Half-Life 2[7] (released in November 2004) are depicted in Figures 6.5 and 6.6. Half-Life 2 also uses shader/effect switching in its LOD system: Figure 6.5 depicts effect switching from a diffuse textured gasoline tank on the side of the truck to an environment-mapped tank representation. This is an example of a LOD switch which can not be handled by geometric smooth-LOD methods; smooth effect switching is made possible through the Unpopping algorithm presented in the next chapter. Figure 6.6 shows part of an object (the sideboards of the truck) appearing out of nowhere, making the hard switch clearly visible even though it takes place far from the player.

Figures 6.3 and 6.4 show popping in the computer game F.E.A.R.[8](released in October 2005). Both show hard switching between different 3D mesh representations. Since the quality of rendering in the game is high, and in these scenes there is nothing to distract the player, the popping is very prominent, even though F.E.A.R. does use LODs sparingly. Figure 6.3 depicts a LOD switch only a short distance from the player, showing that commercial games also do switch LOD early, not late; unfortunately if the switch is done hard, the popping becomes ever more prominent the closer the player is. Unpopping (see next chapter) allows for smooth near LOD-switching.

[6] http://www.wikipedia.org/wiki/Far_Cry

[7] http://www.wikipedia.org/wiki/Half-life_2

[8] http://www.wikipedia.org/w/index.php?title=F.e.a.r.&redirect=yes

Figure 6.1: Discrete LOD hard switching in Far Cry computer game: The small palm tree in the center of the screenshot instantly changes its geometric representation from a decal imposter to a 3D mesh.
To perceive the hard switch dynamically, please visit:
http://www.cg.tuwien.ac.at/research/vr/unpopping/examples/html/fc-1.html.

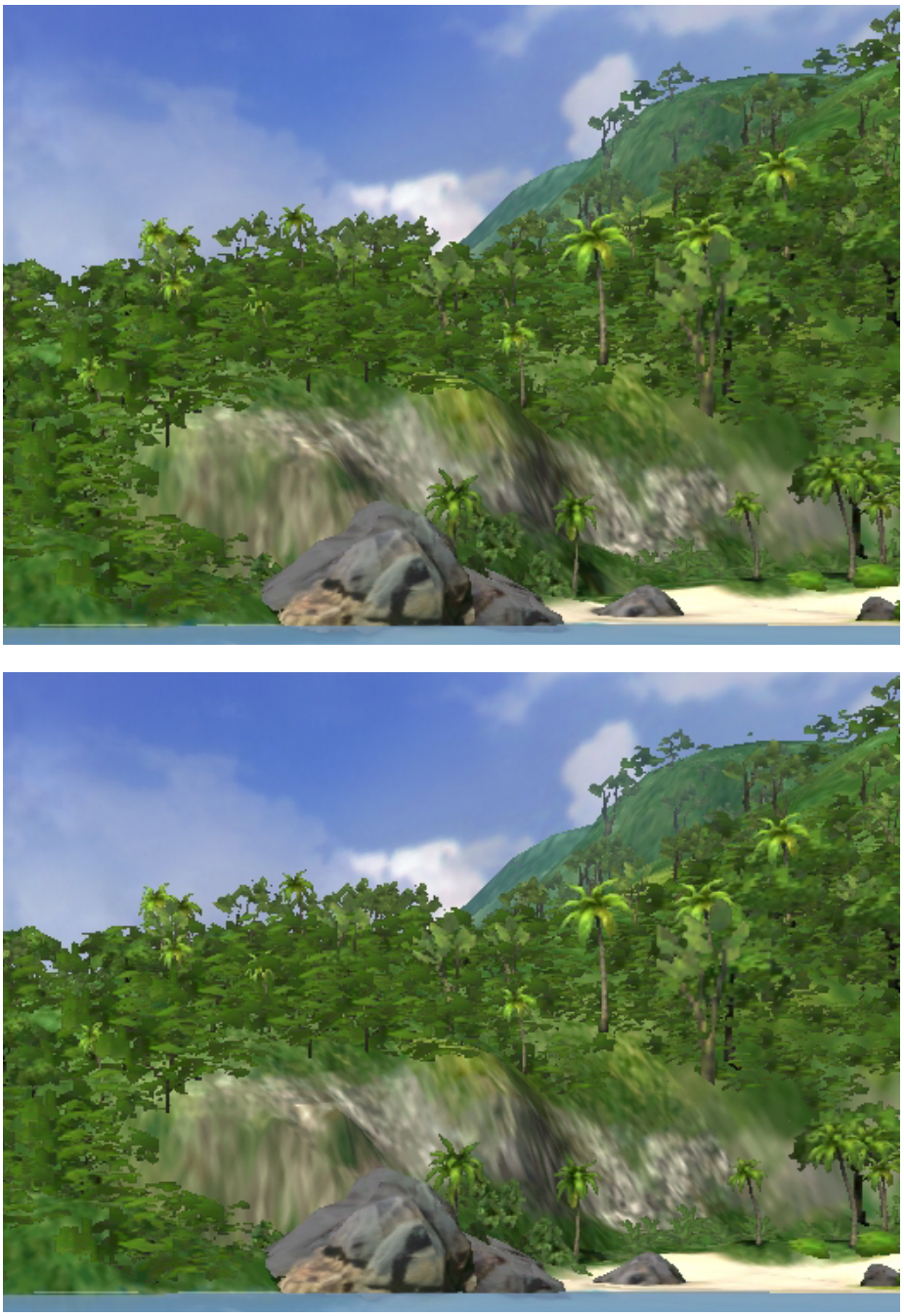


Figure 6.2: Discrete LOD hard switching in Far Cry computer game: Far Cry uses large-scale discrete LODs for its terrain, in addition to the complete vegetation, stones, etc. Switching of the terrain can be observed on the rock face in the center of the screen. This switch is very visible dynamically, since a large part of the rock face changes abruptly; it is however very hard to spot in the screenshots.
Here especially, to perceive the hard switch dynamically, please visit:
http://www.cg.tuwien.ac.at/research/vr/unpopping/examples/html/fc-2.html.

Figure 6.3: Discrete LOD hard switching in F.E.A.R. computer game: Though the differences in the 3D mesh representation are hard to spot here (they occur in the upper arm and leg region), the dead soldier lying on the floor is only a few meters away from the player when the hard switch occurs, making the popping very prominent.
To perceive the hard switch dynamically, please visit:
http://www.cg.tuwien.ac.at/research/vr/unpopping/examples/html/fear-1.html.

Figure 6.4: Discrete LOD hard switching in F.E.A.R. computer game: The 3D mesh representation of the far away dead soldier lying on the floor changes abruptly. The 3D mesh changes by a larger amount than in Figure 6.3 (see for instance the area below the right shoulder pad) and popping is therefore again very prominent, even though it occurs farther away.
To perceive the hard switch dynamically, please visit:
http://www.cg.tuwien.ac.at/research/vr/unpopping/examples/html/fear-2.html.

Figure 6.5: Discrete LOD hard switching in Half-Life 2 computer game: The 3D mesh representation *and* the effects used on the truck are switched abruptly. The geometry switch leads to an abrupt change in lighting; the effect switch on the gasoline tank on the side of the truck switches hard between plain diffuse texturing to environment mapping the tank.
To perceive the hard switch dynamically, please visit:
http://www.cg.tuwien.ac.at/research/vr/unpopping/examples/html/hl2_-_4.html.

Figure 6.6: Discrete LOD hard switching in Half-Life 2 computer game: The 3D mesh representation used on the truck is switched, making the whole side panels over the load floor appear out of nowhere. Parts of an object that appear abruptly out of nowhere are evidently especially problematic; even though the hard switch takes place relatively far from the player, it is still clearly visible and disturbing.
To perceive the hard switch dynamically, please visit:
http://www.cg.tuwien.ac.at/research/vr/unpopping/examples/html/hl2_-_1.html.

Chapter 7

Unpopping

7.1 Introduction

In the previous chapter we have seen that existing approaches to change the discrete LOD representations of an object either suffer from visually disturbing artifacts, defeat the aim of LODs, or do not work at all. It was also addressed that in modern applications, shader LOD[1] is as important as geometric LOD[2] and that shader LOD cannot be addressed by geometry-based LOD techniques such as geomorphing. In addition, geometry-based LOD techniques cannot handle smooth switching between different LOD representations, such as 3D meshes and decal imposters. [3]

In this chapter, we present a practical algorithm for discrete LOD switching that is at the same time visually pleasing, simple to implement and has a very small runtime overhead. The method is based on a new image-space blend formulation of two representations of an object. Only standard, fixed-pipeline graphics hardware features are used. Therefore the method is fast, leads to no conflict with modern hardware acceleration features such as hierarchical depth buffers, and is also suitable for less potent devices such as handhelds.

The algorithm greatly improves the visual quality when changing two discrete LOD representations, and also works for shader LODs.

[1] the use of simpler/faster shaders/effects or rendering techniques for farther away objects

[2] see Figure 6.5 (pg 84)

[3] but see Section 7.7 for a possible combination of the two techniques

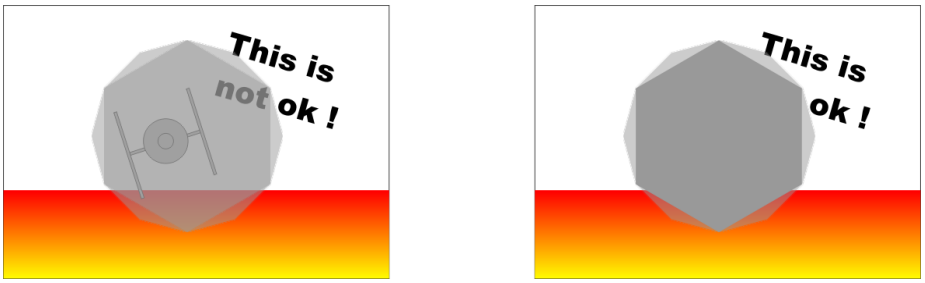


Figure 7.1: A spherical object in transition between two LODs that project to a 6- and 12-sided polygon in screen space respectively. In the background are a spaceship, a planet surface and a text object. Left: Conventional blending leads to semitransparency during the blend, making objects behind the LOD object visible—the player could wrongly see the spaceship. Right: In the new method, one of the LODs is drawn opaquely, solving the transparency problem.

7.2 A New Method for LOD blending

The aim of LOD blending is to stage a smooth transition between two level-of-detail representations, called LOD_1 and LOD_2. The transition should take place for a parameter t, depending for instance on time or distance, running from 0 to 1. The problem then is how to render the object as a blend between LOD_1 and LOD_2 at any parameter value t between 0 and 1.

The traditional interpretation of alpha-blending suggests a blend of $1-t$ times LOD_1 (usually implemented by setting the transparency value of the object to $1-t$), and t times LOD_2. However, this interpretation is unsuited for LOD blending in practice because during the blend both LOD representations would be rendered semitransparently over previously rendered objects and the background (see figure 7.1). If the LOD-blend parameter t is based on distance to the eye-point, the user will see a permanently transparent object as long as his distance to the object stays the same. But even if the blend is done over time, there is no sweet spot for the blend time: Either the blend is short enough, then we perceive popping, or it is longer, then the object is perceived to be semitransparent (which is more disturbing than the pop).

What we would need is an object that at least approximately fills the depth buffer correctly before the blend, so that the scene behind does not shine through. The main idea of our new algorithm is to use the LODs themselves for this purpose, which together with the right choice of depth buffer writes and depth comparison gives a continuous LOD transition in image space:

First, from $t=0$ to 0.5, render the current LOD, LOD_1, opaquely and with depth

		Unpopping		$(\alpha, 1\text{-}\alpha)$
		$t \in [0, 0.5[$	$t \in [0.5, 1]$	$t \in [0, 1]$
LOD_1	α	1	$2(1-t)$	t
	z-test	true	true	true
	z-writes	true	false	false
LOD_2	α	$2t$	1	$1-t$
	z-test	true	true	true
	z-writes	false	true	false

Figure 7.2: The table shows render state settings during the LOD blend. The last column shows the render state settings of the conventional $(\alpha, 1\text{-}\alpha)$-blend in comparison.

writes, and "fade in" the new LOD, LOD_2, by rendering it alpha blended without depth writes but depth compares on top of it. Then, when LOD_2 is faded in completely (i.e., both LODs are rendered opaquely atop of each other and are therefore interchangeable), switch roles, rendering LOD_2 opaquely, and "fading out" LOD_1 from $t = 0.5$ to 1. Thus, at any one time during the transition, one of the two objects is rendered opaquely, and will therefore create a valid depth buffer.

Figure 7.2 shows the different render state settings during the blend.

The new algorithm can also be interpreted the following way: the first stage blends from LOD_1 towards the CSG-*union* of the two LODs, whereas the second stage blends from the union towards LOD_2.

7.3 Discussion

In the algorithm presented, the opaque LOD provides the depth buffer content also for the transparent LOD. This leads to the following minor artifacts:

7.3.1 Silhouette Mismatch

First, in areas where the silhouettes of the two LODs do not match exactly, the semitransparent LOD will blend with the rest of the scene. Also, the part of the semitransparent LOD protruding from the opaque LOD will be hidden by a more distant opaque object if it is drawn afterwards. We have found that these effects are negligible for all practical purposes, since the areas were these effects occur are very small and appear and disappear smoothly. If one still wanted to get rid of the latter effect and minimize the former, one simply would

have to draw the opaque LODs first, and the semitransparent LODs back-to-front afterwards (just like conventional semitransparent objects in the scene).

7.3.2 Depth Discontinuity

Second, if a LOD-blended object *intersects* other geometry, there will be a discontinuity in depth space at $t = 0.5$, when the switch between the LOD being rendered into the depth buffer occurs. In this case a discontinuity at the intersection between the object and the scene will occur, because the transition is not smooth in depth space. This is not a problem in practice, because in correctly modeled scenes intersections between objects and the scene do not occur. In any case, the artifacts that occur are much less pronounced than the popping coming from a hard switch, because the screen space area that changes abruptly is only a small part compared to the area affected by a hard LOD switch.

7.3.3 z-Fighting

One other possible problem is "z-fighting", which can occur if two LODs contain very similar polygons in depth space, which at the same time differ significantly in image space (e.g. due to shading, for example if the normal vectors of the two LODs differ by a large amount). If the LODs are modeled by hand, this problem can easily be avoided by the artist. If they are created automatically and the problem appears, one can extend the algorithm to add a small polygon offset to one of the two LODs.

7.4 Practical Application

7.4.1 Transition Time

In practice a very short transition phase (below one second) is sufficient to eliminate popping and provide a smooth transition. This also gives the best performance since only a small number of objects incur the overhead of being drawn with two LODs at any one time.

7.4.2 Animated LODs

The method also works well for *animated models* with LODs—another area where progressive mesh techniques have a hard time providing good visual quality for low-detail meshes. Geomorphed meshes have a tendency of showing effects like foldovers and too-slim body parts when animated. With discrete LODs, on the other hand, disjointed models (even using line rendering) can be used for the lower-detail LOD representations.

7.4.3 Combination with Geomorphing

The algorithm we presented can also be combined with progressive mesh techniques, for instance doing very high resolution progressive mesh rendering of a model in the foreground, and blending to a discrete, disjointed model LOD representation farther away.

7.5 Blending Different Objects

What might be surprising is the extent to which our visual system is willing to accept any transition as long as it is smooth enough. In practice one finds that the two objects being blended are not necessarily restricted to represent levels of details. It is possible to blend two totally different objects, obtaining the effect of a visually smooth transition between them.

7.6 Electronic Material

At the Unpopping webpage[4] you can find videos and screenshots comparing hard switching (artifact: popping), the "standard" $(\alpha, 1\text{-}\alpha)$-blend (artifact: see-through of background; popping, when blend time becomes $\ll 1s$), and our algorithm. Please note that the artifacts are more pronounced during the actual rendering than in the videos, due to the unavoidable blurring/softening done by the video codec. You can also find videos that show the use of the technique in the commercial computer game "FBI Academy" (and screenshots depicting LOD popping in current computer games, as presented in the last chapter).

[4] http://www.cg.tuwien.ac.at/research/vr/unpopping/examples/

7.7 Conclusion

LOD switching is an underestimated problem: Even the most current computer games still show visible popping artifacts, since the obvious approaches to reduce the popping do not work.

In this chapter we have presented a new interpretation of blending which works on current graphics hardware, with all LOD schemes, and integrates well with the rendering pipeline by providing a valid depth buffer. It can be added to all real-time applications with minimal implementation effort.

7.8 Unpopping Screenshots

The following pages present a comparison between hard switching and the Unpopping algorithm on the example of GameTools Project demogame “Have U Seen My Shadow”, which supports built-in switching between hard switching and Unpopping for LOD switching for comparison.

Figure 7.3 shows two consecutive frames with a minimal amount of forward movement between them displaying different representations of the palm tree which is hard switched between the screenshots, which are just one frame apart.

Figure 7.4 shows a series of six screenshots displaying smooth Unpopping-blending based on distance to the camera. One can observe that, even though the two palm tree 3D mesh representations actually differ quite a bit (compare Figure 7.3), the algorithm does a good job of blending smoothly between them.

Figure 7.3: Discrete LOD hard switching on example of palm tree in GameTools Project demogame “Have U Seen My Shadow”: The two geometric representations are switched abruptly, leading to popping.

Figure 7.4: Unpopping LOD smooth switching on example of palm tree in GameTools Project demogame “Have U Seen My Shadow”: The two geometric representations are switched smoothly, avoiding popping artifacts. Note that there is only a minimal forward movement between the two frames.

Figure 7.4: (continued from last page) Unpopping LOD smooth switching on example of palm tree in GameTools Project demogame “Have U Seen My Shadow”: The two geometric representations are switched smoothly, avoiding popping artifacts.

Chapter 8

The European Union GameTools Project

Figure 8.1: GameTools Special Interest Group members.

8.1 Introduction

This chapter deals with a subject closely related to computer graphics research: The GameTools Project, an European Union project that has as its main agenda leading edge real-time computer graphics research in the fields of Geometry/Plants, Visibility and Illumination, the implementation of C++ 3D graphics libraries based on the research results, and consecutively the tech-

nology transfer to companies developing eletronic games or other 3D graphics based software in the form of the libraries and support.

The author of this thesis holds the position of Community Manger in the GameTools Project and this chapter gives an overview over the project in general, his work as Community Manger, things that went right and things that went wrong, and suggestions for the future.

8.2 Abbreviations

This chapter uses the following abbreviations:

Abbreviation	Meaning
CM	Community Manager (position held in the GTP by the author)
EU	European Union
GTP	GameTools Project
LGPL	GNU Lesser General Public License
PC	Project Coordinator
PM	Project Manager
SIG	Special Interest Group
UdG	Universitat de Girona (University of Girona)
WP	Work Package

8.3 The European Union GameTools Project

The "**GameTools Project**" ("**GTP**") is a European Union specific targeted research project funded under Contract number IST-2-004363 in the IST 6th framework programme[1]. It started in September 2004 and is scheduled to run for 33 months, ending in May 2007, so at the time of writing it is in its final stage.

Figure 8.3 (pg 98) from the GameTools contract [CotEC04] shows the GTP consortium members, which consist of 6 universities ("**academic (consortium) members**"), 1 non-profit organization and 5 companies ("**industrial (consortium) members**") from the fields of game development and virtual reality.[2]

[1] http://cordis.europa.eu/ist/

[2] At the time of writing, two of the industrial consortium members, 3Democracy/Sektor4 and PGM Trading/Saber Interactive[3] unfortunately no longer are in the project.

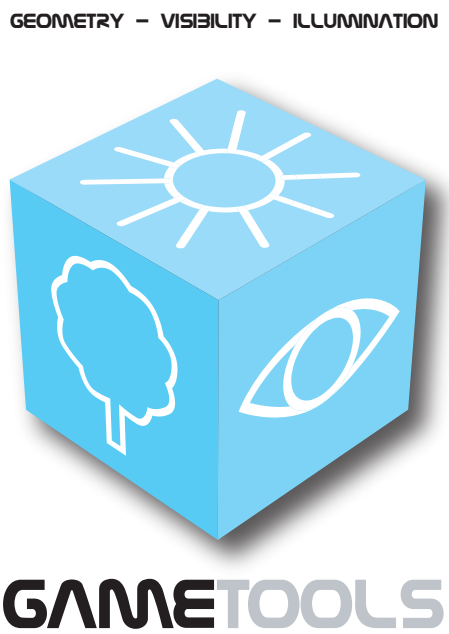


Figure 8.2: The GTP logo. The 3 symbols on the side of the cube represent the three main areas of GTP research: the sun on top for *Illumination & Effects*, the tree on the left for *Geometry & Plants*, and the eye on the right for *Visibility*.

The GTP's main goal is the creation of C++ 3D libraries based on leading edge 3D research from several real-time rendering fields done by the participating academic consortium members; the aim is then to transfer this technology to European companies or companies developing software within the EU, thereby strengthening the European software industry and reducing the brain drain to the US.

The author's job as Community Manager ("**CM**") of the GTP is to advertise GTP technology to companies and inform them about its benefits and the possibility of joining the GTP Special Interest Group ("**SIG**"), to get free preliminary access to GTP technology.

List of Participants

Partic.Role*	Partic. no.	Participant name	Participant short name	Country	Date enter project**	Date exit project**
CO	1	Universitat de Girona	UdG	Spain	1	33
CR	2	Universitat Jaume 1r	UJI	Spain	1	33
CR	3	Universitat Politècnica de València	UPV	Spain	1	33
CR	4	Budapest University of Technology and Economics	BUTE	Hungary	1	33
CR	5	Vienna University of Technology	VUT	Austria	1	33
CR	6	Digital Legends Entertainment	DLE	Spain	1	33
CR	7	Université de Limoges	Unilim	France	1	33
CR	8	PGM Trading ***	PGM TRADING	Austria	1	33
CR	9	Asociación de Investigación de la Industria del Juguete, Conexas y Afines	AIJU	Spain	1	33
CR	10	3Democracy	3Democracy	Denmark	1	33
CR	11	Infowerk softwareentwicklungs	Infowerk	Austria	1	33
CR	12	Gedas Iberia	Gedas	Spain	1	33

*CO = Coordinator
CR = Contractor

** Normally insert “month 1 (start of project)” and “month n (end of project)”
These columns are needed for possible later contract revisions caused by joining/leaving participants

*** The company is currently changing the name from Kernco PGM Trading to PGM Trading

Figure 8.3: GTP Consortium Members (excerpt of the GTP EU contract [CotEC04], pg. 6)

8.3.1 3D API

At the GTP Kickoff meeting in September 2004, it was decided that the main Graphics API that would be targeted would be Direct3D 9. The question which graphics API to officially endorse was brought up by the GTP CM. GTP CM and the industrial consortium members argued for Direct3D 9, since the Microsoft Direct3D graphics API was and continues to be the dominant API used by game developers worldwide, and Direct3D 9 was the – then relatively new – incarnation that supported for the first time Shader Model 3.0, which was a necessary requirement for many planned GTP technologies, especially from the field of real-time global illumination (see next section). The OpenGL API, though pop-

ular with academic researchers, due to its clearer design and greater interface stability from version to version, was not endorsed as an official GTP API; nevertheless, due to this preference, many GTP technologies are also available in OpenGL implementations.

8.3.2 3D Engine(s)

The GTP contract [CotEC04] states that "The tools will be developed around the software that sits at the core of a computer game, the 3D engine.". The main engine that was to be used was already decided before the start of the project: The open source (LGPL[4]) OGRE 3D engine[5] was named the official 3D engine for which all GTP technology would be implemented. OGRE had the benefit of its maturity, its large & active user and developer base, and its relatively modern object oriented C++ design.

Although the GTP EU contract did not call for this, GTP CM pointed out that OGRE was a PC-only 3D engine, with no support for video game consoles. Due to its open source nature and related licensing issues, potential future support for video game consoles (e.g. Sony Playstation 2, Playstation 3; Microsoft Xbox, Xbox 360; Nintendo Gamecube and Wii) seemed also doubtful. A second concern raised was that developing for only one 3D API would increase the risk that the code would turn out to depend on specific OGRE features or even be engrained into the engine itself; this would evidently have hurt the ease with which companies could have integrated GTP technology into their engines (since OGRE is a general purpose, open source, PC-only engine it is extremely rare to see it used in commercial projects, especially in game development). To counter both problem areas the author suggested to endorse a second (commercial) 3D engine with support for video game consoles. As an additional benefit, all customers of the chosen engine could automatically benefit from GTP technology (once they became GTP SIG members), since it would already be integrated in "their" engine. It was unanimously agreed by all GTP consortium members to commit to adding support for a second engine to the GTP agenda.

Several middleware solutions (including Gamebryo[6], RenderWare[7], Shark 3D[8] and Unreal 2 & 3[9]) were evaluated by the author.

[4] http://www.gnu.org/licenses/lgpl.html
[5] http://www.ogre3d.org/
[6] http://www.gamebryo.com/
[7] http://en.wikipedia.org/wiki/RenderWare
[8] http://www.shark3d.com/
[9] http://www.unrealtechnology.com/

From this two prime candidates emerged: **RenderWare** by UK based Criterion Software [10] and **Shark 3D** by Germany based middleware supplier Spinor [11]. Both companies were developing C++ 3D middleware solutions within the EU, and both RenderWare and Shark 3D supported a wide range of video game consoles. Talks with both companies followed, and while Renderware was the more widely used product, at the time there were persistent rumors that Electronic Arts (then and now the largest developer and publisher of electronic games worldwide), who had bought Criterion in July 2004, was going to cease licensing it to third party developers, instead turning it into a pure in-house tool. This, together with the fact that Spinor seemed more inclined and able (due to their smaller size) to cooperate flexibly and efficiently with the GTP, indicated Shark 3D to be the overall better choice.

Shark 3D was consecutively tested in the CG23 game development lab course[12] by two teams of students to see how well it worked in practice. The test went well and, following the suggestion of GTP CM, Shark 3D was finally picked to be the second engine to be supported by the GTP.

The decision against RenderWare proved to be a wise one, when Electronic Arts consecutively did proceed to turn RenderWare into an in-house tool.

The following section gives an overview of the real-time research areas GTP technology was researched and developed in.

8.4 GTP Research Areas

Research and implementation of GTP results is done in the following real-time research areas, represented within the GTP contract in the work package ("**WP**") 3, 4 and 5 respectively:

8.4.1 Visibility (WP3)

Visibility research was carried out at Vienna University of Technology ("**VUT**").

The areas researched and implemented included:

1. Online ("on-the-fly") visibility culling in image space
2. Precomputed Visibility

[10] http://www.criteriongames.com/

[11] http://www.spinor.com/

[12] regularly held each year for several years running by GTP core consortium member VUT

(a) Quasi-exact

(b) Progressive Sampling-based

8.4.2 Geometry and Plants (WP4)

Geometry and Plant research was carried out at Universitat Jaume 1r ("**UJI**"), Universitat Politcnica de Valncia ("**UPV**"), Universit de Limoges ("**Unilim**"), Universitat de Girona ("**UdG**") and Vienna University of Technology.

The areas researched and implemented included:

1. Continuous Level of Detail for
   (a) General Geometry
   (b) Trees and Plants
2. Offline Level of Detail Creation, with simplification metric being
   (a) Geometric
   (b) Image Based
3. Billboard Cloud Trees
4. Smooth Switching of Discrete Levels of Detail[13]

8.4.3 Global Illumination and Effects (WP5)

Global Illumination and Effects research was done at Budapest University of Technology and Economics ("**BUTE**"), Vienna University of Technology, Universitat de Girona and Universit de Limoges.

The areas researched and implemented included:

1. Global Illumination
   (a) Raytrace Effects: Environment Depth Maps for Reflections and Refractions[14]
   (b) Exact GPU Raytracing
   (c) Indirect Illumination
   (d) Ambient Occlusion

[13] author's work; see Chapter 7.

[14] Raytrace Effects employ approximate raytracing on the GPU to give much improved quality for reflection and refraction effects applied to arbitrarily shaped geometry, especially if the reflected/refracted objects are near the reflector/refractor, i.e. without a far-field assumption.

2. Shadow Maps
   (a) Shadow Map Reparametrization
   (b) Virtual Tiled Shadow Maps[15]
3. Effects
4. Depth Imposters[16]
   (a) Single (e.g. explosions)
   (b) Clustered (e.g. fog)
5. Metallic Surfaces
6. Depth of Field
7. Glow
8. Rain

8.5 GTP Special Interest Group

The GTP Special Interest Group ("**GTP SIG**") is defined in the GTP EU contract as the legal instrument for the technology transfer from the GameTools Project to companies developing software within Europe.

8.5.1 SIG Membership Agreement

Although the primary goal of the GTP is to give software companies developing in Europe free access to GTP technology, this does not mean that the GTP libraries are freeware.

The reason why the GTP libraries cannot be freeware is the restriction imposed by the GTP goal of boosting the European software industry (while at the same time aiming to exploit the technology economically): Only companies employing GTP technology for software development within the EU should be allowed to use it for free (**EU Focus**).
This created the need for a license agreement between the GTP and those companies.

[15] author's work; see Chapters 3,4 and 5.

[16] Depth Imposters solve the problem of conventional flat, semitransparent decals intersecting with opaque scene geometry, which normally shatters their volumetric illusion. Their more technical name is "Spherical Billboards".

Since the GTP SIG had been created for companies to join to get access to GTP technology together with support, it was natural to integrate the software license into the GTP SIG membership, leading naturally to the concept of the GTP SIG Membership Agreement.

Though the restriction of EU Focus described above in itself required a legal agreement of some sort, there were also other interests by the GTP and its consortium members which had to be fulfilled by it. These were established by the author through communication with other GTP consortium members, and then formally categorized and defined as follows:

1. SIG Membership Official (together with use of SIG members name and logo to this respect): This allows e.g. for SIG members to be listed on the GTP webpage together with their logo. This has a positive effect not only for the SIG member (free advertising, potential for networking with other SIG members), but also the GTP (since the more companies have already joined, the easier new companies will also decide to join[17]) and the EU, since the public can see on the GTP webpage that the project is fulfilling its goals, i.e. the EU money is well spent.

2. Credit: This is similar to "SIG Membership Official" above, only with regard to the fact that SIG members are required to give credit to the GTP when they ship a product which uses GTP technology: Same as with many libraries, SIG members are required to state that they are using GameTools technology in their product when shipping it. To this effect, the GTP logo with the text "based on technology from the GameTools Project (www.gametools.org)" below it must be included in either the product's manual or credits.

3. Scientific Results: Since the GTP represents leading edge research, the participating universities (academic consortium members) need to have protection against their technology being used by someone else to produce scientific results and publications. SIG members get full access to optimized sourcecode (and even support), something which is not normally done in the academic community, since no researcher or research team wants to lose the edge he/they might have developed in a particular field due to being able to build on the software base that was already implemented for previous research in the same field.

4. Library Creation: Since the GTP is supplying its libraries for free to SIG members, it does not want to allow SIG members to create and/or sell libraries based on GTP technology. This is also related to another GTP

[17] One could say that most people have a tendency to be, in this regard, bosonic by nature.

goal, that of exploiting the results after the end of the project in a commercial manner; one way to do this is evidently by selling the GTP libraries (to non-SIG members in the EU or customers outside of the EU, e.g. in the US or Japan).

GTP CM authored several drafts of the agreement, being in contact with GTP Project Coordinator, Mateu Sbert, and GTP Project Manager, Jordi Palau. He also included feedback from a representative medium sized game development company which takes IP infringement issues seriously, under what conditions they would be interested in becoming a GTP SIG member, respectively what would put them off. After a decision was reached on how to address potential intellectual property infringement issues (copyright, patents, etc) in the SIG draft, the 1.0β version was emailed to IPR Helpdesk for a final checkup. IPR Helpdesk[18] is an organization funded by the EU to supply legal assistance regarding intellectual property issues to EU projects. The relatively minor issues the IPR Helpdesk lawyers highlighted where integrated into the final 1.0 version of the document by GTP CM, which was consecutively accepted by all GTP consortium members.

The agreement was voluntarily kept as concise and liberal as possible, as to make the task of checking it by potential SIG members as hassle free as possible, also aiming to keep legal costs on the side of a potential SIG member as low as possible, should he decide to involve a lawyer.

Figure 8.21 (pg 142) shows the 4 pages of the current GTP SIG Membership Agreement (v1.1).

The agreement consists of a general and a specific part, where the General Part contains standard game middleware software licensing stuff, such as Confidentiality, Copyright & Ownership, etc. The Special Part establishes the restrictions described above and the right of the GTP SIG member to use GTP technology for any purpose, condition to these restrictions. See also the GTP webpage section about the SIG membership agreement[19].

8.5.2 GameTools Evaluation License ?

The author has considered offering an “evaluation license”, to enable companies to check the libraries and documentation before they become full SIG members, but decided against it, for two reasons:

[18] http://www.ipr-helpdesk.org/

[19] http://www.gametools.org/html/sig_membership_agreement.html

First, he knows from experience that evaluation versions of commercial libraries often sits on an employees hard drive for a long time, since no one finds the time to actually do the evaluation; this is especially true for game development companies, which have some of the tightest deadlines in the industry and therefore very little time for such tasks. This would have meant that in many cases companies would not have become SIG members, simply due to this fact.

And second, since it was clear from the start that it would, in any case take quite some time for the GTP libraries to become full fledged and mature enough to stand up to scrutiny.[20] This would have meant that the author as GTP CM could have started his job of advertising the GTP to potential SIG members even later, to not risk scaring companies away by a too early version of the GTP libraries.

8.5.3 GTP SIG Questionnaires

The author designed a GTP SIG questionnaire[21] which he mailed to all existing SIG members on 28 02 2007. The questionnaire contained questions about:

- The usefulness of GTP technology to the companies.
- The companies current and planned future use of GTP technology.
- The development libraries & tools etc employed by the companies.
- Interest in Participating in GameTools events.
- General interest in respectively specific interest to participate from the start in "GameTools 2"[22]

8.5.4 SIG Members

The following lists the current 20 GTP SIG members, together with their logo and webpage, where you can find up-to-date information about each respective company.

[20] Although in the end it turned out to take even longer than the author expected.

[21] http://www.gametools.org/GTP_SIG_Questionnaire.rtf

[22] the GTP follow-up project

Animante Baleares

http://www.animante.com/

Sproing

http://www.sproing.com/

Brainstorm Multimedia

http://www.brainstorm.es/

Invictus

http://www.invictus-games.com/

TAB-Austria

http://www.tab.at/tab/en/

VIS

http://www.vis-games.de/

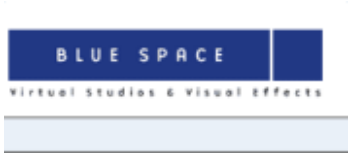
bouncing bytes

http://www.bouncingbytes.at/

Elektra Project

http://www.elektra-project.org/

Blue Space Media GmbH



http://www.blue-space.de/

Bogengang GmbH

http://www.bogengang.de/

Candella Software LTD

http://www.candellasoftware.com/

Framework Studios

http://www.framework-studios.com/

Over the Edge - 3D Engine

http://www.otee.dk/

Tragnarion

http://www.tragnarion.com/

Carintia

http://www.carintia.es/

Vizrt

http://www.vizrt.com/

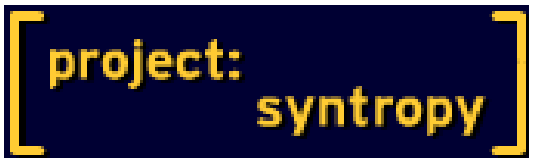
Nemesys

http://www.nemesys.hu/

Phoenix interactive

http://phoenix-i.fr/

Project: syntropy



http://www.project-syntropy.de/

vr3 virtual production

http://www.vr3.de/

8.5.5 GTP Software Repository

Following an evaluation by the author, Subversion[23,24], was finally chosen over alternatives such as CVS[25], as the GameTools software repository solution. The decision was taken through a poll in the internal GTP forum[26] open to all GTP consortium members.

The author then proceeded to create several iterations of repository schemes, discussing them via email with GTP WP3 members Michael Wimmer, Jiří Bittner and Oliver Mattausch.

The final GTP software repository scheme is depicted in figure 8.14 (pg 136); The following is the description of the structure, sent out through email to all GTP members on 22 02 2006:

```
1) The toplevel dir contains 5 subdirs:

GTP
GTP-Internal
NonGTP
obsolete
OGRE

GTP ... contains all code that the GTP is responsible
for (copyright infringement etc)

GTP-Internal ... contains code that is for internal GTP use
only; GTP SIG members will have no access to this directory; you
are not required to put anything in here, but if you have GTP
internal documents etc, which you want to store in the
repository: Here is the place.

NonGTP ... Non-GTP libraries, such as boost, devil etc

obsolete ... nomen est omen: Old stuff which you might want to
keep for reference; we stored our old visibility projects
structure here.

OGRE ... Since OGRE is partially our code and partially external
code, it got its own directory; the OGRE version stored here is
1.07. If you are still using an older OGRE version, please
```

[23] http://subversion.tigris.org/

[24] {http://www.wikipedia.org/wiki/Subversion_%28software%29}

[25] http://www.wikipedia.org/wiki/Concurrent_Versions_System

[26] http://www.gametools.org/egroupware/login.php

```
upgrade to this version. You must not check in your own OGRE
version, but must use the existing one ! Changes made to the
OGRE engine should go into \OGRE\trunk\ogre_changes. If a file
you changed already exists, you are responsible for merging it
with your changes.

2) Each subdir contains the standard SVN folders:

trunk, branch, and tags

The "trunk" folder is for the current development branch of the
tree, meaning that code developments should only be made in the
"trunk" folder.

The tags folder is to check in milestones, stable versions, etc.
The branch folders is for development that is done away from the
main development in trunk, but which will later be merged with
the sourcecode in trunk again.
```

The following is an excerpt from an email by the author to all GTP WP leaders, sent on 21 09 2006[27] suggesting additional improvements to the content in the software repository (slightly reformatted):

```
1) Each project must include a file "README.txt" at its root
level, which explains

a) The effect/technology provided (including rough status of
completition)

b) Benefits of the new algorithm(s)

c) How to get the project to build / run (OGRE paths etc; other
requirements)

d) Any other additional information that comes to mind, that
might be useful to someone who is not a computer graphics
researcher to get interested in this GTP part and to help him
understand what we are doing and how he could apply it in a game
/ 3D application.

2) Each project should have a stable executable, which anyone
can start to get a quick look at what the project does. This is
your easiest way to get people to actually use your effect - so
```

[27] Incidentally the author's birthday :-).

```
try to make this impressive (for instance, try to avoid
rendering into a 640x480 window or smaller by default, if a
GF6800 or higher is fast enough for higher resolutions).

3) Every executable should adhere to the following standard
(Displaying a helpscreen when F1 is pressed is by far the most
important):

F1: Help screen(s); displays an in-game help screen or help
screens, giving short information on all user input
possibilities (keyboard, mouse) and some         information
about the demo (1-2 lines). This does not have to be fancy, but
should simply output some text giving information on how to use
the program.

ESC: Exit game (without query)

P: Pause/Unpause the game (toggle)

W, Keypad 5, Cursor Up: Forward movement

A, Keypad 1: Strafe left
D, Keypad 3: Strafe right
S, Keypad 2, Cursor down: Backward movement
Cursor Left: Turn left
Cursor Right: Turn right
Space, J: Jump
Alt, Return, Enter, I, Right Mouse Button: Interact
Left Mouse Button, Ctrl: Fire (in the broadest sense)

Mouse movement: standard mouselook (mouse up ... look up, mouse
left ... look left, etc)

The README.txt file is the most important measure - please see
that every project has someone responsible to add this file and
that it is done quickly ! Stable executables are a close second,
together with F1 displaying a help screen.
```

The suggestions were consecutively implemented to a large degree, improving the overall accessibility of the GTP content in the repository.

8.6 GTP Support Forum

In his function as GTP CM, the author created a GTP support forum[28], to allow the GTP WPs and Spinor to efficiently give support to GTP SIG members, GTP demogame teams, and other student teams (such as the Shark CHC team at VUT). The author decided to use a phpBB forum[29] hosted at VUT, with a GTP design based on the phpBB ChunkStyle template.

The forum was in the end mostly used for support of the Shark CHC port and to a small extent for the GTP demogames. The author made sure that every GTP SIG member was aware of the forum's existence and had a valid account there, by instructing every new SIG member to create an account for the GameTools support forums[30], and consecutively sending the GTP Software Repository (see Section 8.5.5) access information through a private forum message; despite that, GTP SIG members did not use the GTP SIG forum. The returned GTP questionnaires seem to indicate that the reason for this is, that unfortunately most of them had not had found the time to look at GTP technology thoroughly enough to post support questions to the forum.

Figure 8.15 (pg 137) shows the GTP support forum start page.

8.7 GTP Advertising

8.7.1 GTP logo

The original GTP logo is depicted on the left of Figure 8.4: The author in his function as GTP CM criticized the original logo design, because it did not indicate what the GameTools Project was about, but on the contrary, through the dominant wrench depicted in the back, emphasized the wrong impression that GTP was about some sort of game development "tools".[31]

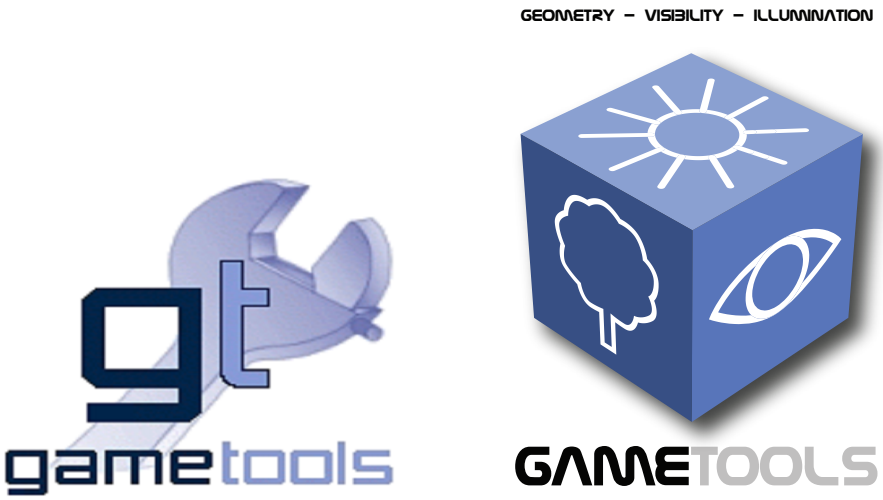
Evidently the term "tools" also already appeared in the name of the project, which was unfortunate, since most people from the industry think of e.g. 3D editors when they hear about "tools for game development", and not 3D libraries.

[28] http://forums.gametools.org/

[29] http://www.phpbb.com/

[30] followingthenamingconvention"SIG"+<companyname>,foreasieridentification

[31] in addition the author thought that the wide, backward looking "g" did not send the right "next generation graphics libraries" message, especially when the wrench, who supplied some upward motion, was removed.



(a) old logo **(b)** new logo

Figure 8.4: Old GTP logo on the left, which was replaced at the start of the project with the design of the author on the right (depicted in the GTP webpage color scheme).

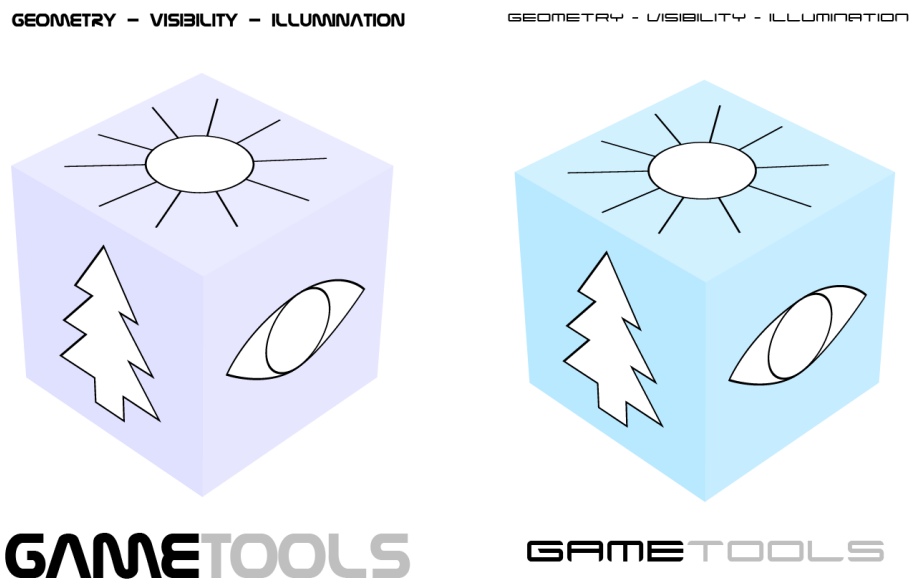
Since misconceptions about the aim of the GTP would make the job of advertising harder, and the name of the project could not be changed any more (due to being in the EU contract), the author lobbied for a change of the GTP logo.

UdG then hired an external designer, who, alas and despite the author's best efforts, did only produce designs which seemed to be built on the concept of children's toys or gambling; again this was, in the author's opinion, not the image the GTP, having as its aim the creation of professional libraries for a high-pressure industry which was a far cry from childrens' toys or gambling, wanted to project.

So finally the author created the design based on a "GTP cube" (representing technicality), whose sides showed symbols representing the 3 major areas of research within the GTP: the sun for illumination/effects, the eye for visibility and the tree for geometry/plants; he included the text "geometry - visibility - illumination" at the top of the logo, to supply additional information about the GTP.

After some internal struggle, and several design iterations, the design was finally accepted as the new, official GTP logo. Figure 8.2 shows the final logo in the "ice blue cube" color scheme, figure 8.6 shows an example of the logo used on the GTP CM business card.

Figure 8.4 compares the old with the new logo (in the GTP webpage color



(a) proto logo 1 **(b)** proto logo 2

Figure 8.5: Two earlier designs for the new GTP logo.

scheme). Figure 8.5 shows two earlier designs from which the final GTP logo emerged.

Figure 8.6: Example of GTP logo use on GTP CM business card. The business card also incorporates information about the runtime of the project (2004-2007) into its design.

8.7.2 GTP webpage

Since the author did not have access to an advertising budget, the GameTools webpage was the obvious and most important advertising instrument to attract

companies to join the GTP Special Interest Group (see Section 8.5).

Unfortunately the original webpage design was based around a small fixed-size (HTML) content area window, which was not well suited to display e.g. a list of GTP demo videos or more than a few sentences of information. It also had some very obvious graphical errors (e.g. non antialiased parts of the border slices).[32] This combined gave it an unprofessional and dated look.

The author then tried to work with the persons responsible for webpage design and maintenance at UdG. Unfortunately, despite a large effort on the author's side, response times were very slow, and the author therefore agreed to take over webpage maintenance as time permitted, to speed up the improvement of the webpage to become an advertising tool.

The webpage remained in the author's care until October 2006, when a conflict between GTP PC Mateu Sbert and GTP CM over who should decide what content is put on the webpage[33] led to the webpage maintenance being transferred back to UdG.

During the time the author maintained the webpage, he has expanded it considerably, introducing all major sections, in addition to overhauling it several times and continuously adding content whenever he got hold of videos/images/etc which helped advertise the GTP to companies. No major changes have occured after the transfer back to UdG and to this day the largest part of the content on the webpage (with the exception of the videos, which where created by the different GTP WPs) comes from the author.

Figure 8.7 depicts the current GTP start page; figure 8.8 and 8.9 show the webpage news and video/demo section.

8.7.3 Game Developer Conference Europe 2005

The author attended the Game Developer Conference Europe 2005 (GDCE 05), held from the 30 08 2005 until 01 09 2005 in London, to test the

[32] In addition to very basic problems, such as the use of Microsoft BMP graphics file format for images.

[33] The main point of the conflict was about whether all GTP members should have the right to publish whatever they like on the webpage, or whether this should be filtered to fit the purpose of advertising the GTP. The webpage had been developed as an advertising tool by the author for one-and-a-half years at this point, and the sudden change to turn the webpage from an advertising tool to a blog where everyone could post what he saw fit came therefore very abrupt and unexpected. The author also had managed to get free GTP advertising on several important game development webpages at this point (see Section 8.7.6), following the launch of the first GTP demogame (see Section 8.8) so he managed to postpone the switch at least until the news value of the announcement had faded.

waters, preliminary advertise the GameTools Project to European game developers and also to learn first hand about future trends in game development. Preliminary, because at this time the GTP was still in an early stage, and not yet in the position to allow SIG members8.5 (pg 102) in, since there was not enough available technology. Opening the GTP SIG to early, i.e. with little to no useable content, would have risked getting bad word of mouth by disappointing the companies who had gone through the process of evaluating whether to join (see Section 8.9 (pg 124)).

The author talked to a lot of game developers and people from neighboring industries, swapping contact information and handing out GTP information cards.

Due to the large number of approximately 700 attendees it was of course impossible to reach all of them through personal conversation. The author concluded that one would have to either invest in a stand or a flyer being included in every attendee bag to get the maximum benefit out of such an event. Note however that the cost of such measures are considerably higher than simply attending, and therefore have to be weighed against e.g. advertising in a magazine.

Note that while giving a talk about the GTP might look like the perfect free advertising opportunity, in the author's experience, alas only a very small number of people at such conferences attend such specialized talks by unknown speakers, limiting the potential audience considerably.[34]

He also talked to British electronic gaming journalists, advertising the GTP; in this context it became obvious that, while the message was received in a friendly manner, the question was raised, why there were no British companies or universities involved in the GTP as core consortium members. There is a lesson to be learned here, in that a project such as the GTP should in the future aim to include companies and universities from the UK, since it is by far the country with the largest game development industry in Europe.

8.7.4 GTP Demo Games

The GTP demo games, developed by students of Vienna University of Technology, have their own section (8.8) below, since they also serve other purposes, apart from being an advertising instrument.

[34] down to near zero, if the talk is scheduled in parallel to a big presentation, such as by Sony or Microsoft

8.7.5 Eurographics 2006

At Eurographics 2006, Europe's premium computer graphics conference, held in Vienna from the 4thto the 8thof September 2006, the GameTools Project was presented at its own booth, showcasing live GTP technology demos to an interested audience. GTP information flyers and contact information in the form of business cards (see Figure 8.6) were also made available.

For further exposure the GTP, together with NVidia[35], also sponsored the Graphics meets Games competition at Eurographics 2006[36]. Figures 8.18 (pg 140) and 8.19 (pg 141) show GTP advertising at the Graphics meets Games competition at Eurographics 2006.

The Eurographics news was consecutively presented on the newly introduced News section of the GTP webpage[37] (see Figure 8.8 (pg 131)).

8.7.6 gamasutra.com et al

Although there still alas was not as much content in the GTP Software Repository as the author would have liked, after Eurographics 2006, the author decided that it was nevertheless time to finally officially open the GTP Special Interest Group. The first GTP demogame, "Jungle Rumble", had been finished, and this together with the SIG opening would hopefully create enough newsworthiness to be listed on the respective game development news sites on the web.[38]

The author created a three tiered news message, a one-liner, short- and long version, each aiming to entice the reader to learn more, and submitted it to gamedev.net[39], devmaster.net[40] and gamasutra.com[41]; these sites are important, not only because they are themselves being frequented by game developers, but also because the news they accept is also automatically listed on other, less specific tech news sites.

In addition the news was also posted to gamedev.org[42], a forum visited by many game developers.

[35] http://developer.nvidia.com/

[36] http://www.cg.tuwien.ac.at/events/EG06/graphicsmeetsgames.php

[37] http://www.gametools.org/html/news.html

[38] E.g. one site the author submitted to listed 30 news items with pending reviews for the day; from these, only 2 actually appeared on the page.

[39] http://www.gamedev.net

[40] http://www.devmaster.net

[41] http://www.gamasutra.com

[42] http://www.gamedev.org/forum/index.php?board=7.0

The GTP SIG opening & demogame launch news was accepted on all sites[43],[44],[45] and in the end lead to 8 new companies joining the GTP SIG.

8.7.7 Resfest 2006

In November 2006 the GTP was present at the international Multimedia Resfest at the Museumsquartier at the heart of Vienna. Resfest[46] is an international festival for Independent films and multimedia which tours major cities the whole world over, drawing a crowd from the creative industry.

Figure 8.20 shows the poster used to advertise the GTP at the Resfest 2006.

8.7.8 GTP Game Developer Magazine Ad

In February 2006, after GTP CM evaluating several different options, GTP CM together with the GTP CFO at VUT (WP3 leader Michael Wimmer) decided to advertise the GameTools Project in Game Developer magazine[47].

Game Developer magazine is the worlds premium publication aimed specifically at game developers, being read by more game developers worldwide than any other publication. Other venues considered were for instance the German game development publications Game Face[48] and /Gamestar/dev[49] and renowned UK game magazine "EDGE"[50].

Game Developer magazine in the end won over EDGE, presenting the best value for money, with a newly created advertising section in the center of the magazine and due to its large international appeal in the main target group.

Figure 8.10 depicts the ad, which was designed to be seen on a dark grey background, taking up a quarter of a page, together with 3 other adverts from the game industry.

[43] http://www.gamedev.net/community/forums/topic.asp?topic_id=416808
[44] http://www.devmaster.net/news/index.php?storyid=1201
[45] http://www.gamasutra.com/php-bin/news_index.php?story=11077
[46] http://www.resfest.com
[47] http://www.gdmag.com
[48] http://www.game-face.de
[49] http://www.gamestar.de/dev/
[50] http://www.edge-online.co.uk/

8.8 GTP Demo Games

The author supervised the creation of 3 GameTools demogames by teams of students, which were created to showcase GTP technology in game environments.

8.8.1 Jungle Rumble

Jungle Rumble was the first GTP demogame to be finished. It is a comic-style fun driving & shooting game, where the player controls a giraffe sitting in a bouncy, nutshell-like car, and has to battle it out with a host of fun weapons with several motorized zebras.

The games consists of several separate distinct levels, all having their own colorful theme, such as volcano, alien crash site or sugar island. Each level is set on an island surrounded by water, where the terrain of the island can be demolished by the weapons at the players and his opponents disposal. The player weapons include: miniguns, sci-fi blasters, bombs and a flamethrower and ice-gun.

The game showcases the GameTools effects of Depth Imposters (see Section 4 (pg 102)) and Raytrace Effects (see Section 1a (pg 101)) very well: Depth Imposters are employed for large explosions, flames in the volcano level, the ice-gun and especially the flame thrower (where they are also combined with a heat blur effect). Raytrace Effects are used on several larger items in the game. Raytrace Effects are a relatively costly effect, and they have been cleverly included into the engine to support several objects using the effect, including interreflections and -refractions, at high framerates.

Jungle Rumble is fun to play and employs freely downloadable music from several bands[51] to increase the gameplay experience.

Figure 8.11 (pg 133) shows screenshots from Jungle Rumble. The game can be downloaded from its GTP webpage subpage[52], where you can also find more screenshots.

8.8.2 Have U Seen My Shadow

“Have U Seen My Shadow” was the the last GTP demogame to be started. Have U Seen My Shadow is a small 3D adventure game set on a caribbean

[51] with the bands permission, of course

[52] http://www.gametools.org/html/jungle_rumble.html

island, following along the lines of such classics as "Monkey Island"[53]. The basic story evolves around the idea that the player has lost his shadow, and must try to find it again.

The Have U Seen My Shadow demogame team initially consisted only of a single developer, which was after some time joined by two artists, which proceeded to produce high quality 3D art for the game. The programmer and team leader in the team had only minimal game engine design and implementation experience at the beginning of the demogame project, and also had to work besides his studies. Have U Seen My Shadow has therefore not been coming along as fast as initially planned. Despite this, also due to the author sharing his experience in engine design with the Have U Seen My Shadow team, the game is looking nice and making progress.

Figure 8.12 (pg 134) shows screenshots from Have U Seen My Shadow. Note that the light direction in the screenshot is orthogonal to the view direction, leading to the optimal LiSPSM case; shadow quality therefore reverts to uniform shadow mapping when the camera is turned to look into the light. Vegetation and terrain use the author's Unpopping algorithm to blend smoothly between discrete LOD representations.

8.8.3 Penta-G

Penta-G[54] was the GTP demogame launched first and with the largest team, having 3 team members, 2 of which were programmers from the start. Penta-G was to planned to be a small first person shooter. The core programmers, contrary to the other two GTP demogame teams, claimed prior experience in having successfully implemented game engines (for fun, i.e. non-commercial); this led to high expectations on the side of the author, and he accordingly invested the most time to this team, managing to add another two artists to it, and trying to steer it into the right direction[55], according to the prime guideline for all GTP demogames: "Visual quality should be high & display GTP libraries capabilities, gameplay should be engaging. Small & simple, but excellent is the

[53] http://www.wikipedia.org/wiki/Monkey_Island_%28series%29

[54] from the greek "penta", five

[55] Apart from the obvious GTP focus, another pitfall, especially in this team, was overambitiousness. The initial game concept presented to the author was overambitious according to his own experience, and could not only have not been finished on time, but ran a very high risk of never being finished at all, a fate that many university game projects suffer. The time it has taken Penta-G to come this far proves the author right.

goal :-)"[56]

Unfortunately the core team always had a hard time understanding that this project, for which they expected to be graded for the amount of 12 hours of work per week each, was not their private playground for programming a game for fun, but aimed at "display[ing] [the] GTP libraries capabilities" in the best way possible. Their attitude towards the GTP libraries, which naturally were at the time at an early stage, of which they had been informed before the start of the demogame project, contrasts interestingly with the Jungle Rumble team above: While the Penta-G team insisted that GTP Raytrace Effects were too slow and gave too poor a quality to be used in practice, the Jungle Rumble team had already implemented them well into their engine and continued to improve their use until the game was finished.[57]

The "prior experience" did alas also not manifest itself, especially the one programmer who also started his master thesis in computer graphics under the author, and of whom the author unfortunately had to discover that his own self assessment with regards to his programming skills was in crass contrast to reality, when he was forced to take over the research implementation and do the implementation of QVSM himself[58], due to the students silent dropping out from his master thesis[59]

There have been a lot of other problems with the Penta-G team during the time the author oversaw its development, such as the team not understanding that enemies and the base level and opponent design should evidently be created first[60], without a high level of detail, to find out problem areas early on, and allow for changes to be implemented fast by the artists, but the game is already taking up too much space in this thesis, so the author will not go into any more detail.

In the meantime a lot of people have worked on Penta-G, and it looks like it is

[56] from an email of the author to the Penta-G team at the start of the Penta-G GTP demogame project, on 18 10 2005.

[57] Interestingly enough, they also insisted that the existing, proven to work LiSPSM [WS06] code accompanying the book "Shader X4 - Advanced Rendering Techniques" could not be integrated into their engine, and that the GTP Coherent Hierarchical Culling algorithm (CHC) [BWPP04] did not work in practice. They have held to the latter belief until the time of this writing, even stating it in their written work about the project.

[58] in a conversation before that the student had tried to convince the author that the QVSM algorithm could not work in principle, alleging that hardware occlusion queries could not be used in the way the author intended.

[59] of whom he had initially stressed that he absolutely needed to have it finished in the summer of 2006. To the author's knowledge he has not finished his studies to this day, one year later.

[60] see e.g. Half-Life 2 for a high profile example that followed this paradigm that seems very obvious to the author.

finally nearing completion. Unfortunately the interesting visual design concept of the author, based on a 5-sided symmetry has been watered down to look like a Shareware commercial first person shooter rip-off[61] This is glossed over by a standard post processing glow effect and well executed 2D control elements.

The problems with the underlying implementation alas persist to this day, and the program is still bugged by unexplainably low framerate, badly working CHC implementation, and stuttering, which the team still seems to have troubles to fix.

Figure 8.13 (pg 135) shows screenshots from an earlier version of the game, still based ont the author's design; the author is convinced that, had the team been willing, the demogame could have been finished from this point on within one month.

8.9 Why GTP Technology is Not Free

8.9.1 Free as in No Cost ?

The concept of the GameTools Project seems to imply that the technology created within the project is "free", where "free" is taken to mean "incurring no costs". It is the author's impression, that many GTP members, including the GTP PC, adhere to this view.

This section discusses why no software library, and especially the libraries & technologies created within the GTP, can ever be "free" in that sense.

8.9.2 Public Domain Libraries

Let us first start by looking at a software library that is in the public domain, meaning that there are no restrictions of any kind linked to using it, and anybody is allowed to do with it whatever he likes.

Certainly public domain libraries can be seen as being "free" in the most extreme sense of the word, so it might come as a surprise, that using them in a commercial software project does not come without a price; the costs incurred by public domain libraries for a company are as follows, in temporal order:

[61] at least it has also incorporated good ideas from other games created at VUT, such as the rolling polyhedron with a light inside of it, shining through holes in its sides, taken from the VUT student game "Cubophobia" (`http://www.cubophobia.at.tt/`) `http://stud4.tuwien.ac.at/ e0126452/cubophobia/screenshots/full7.jpg`, created during the course of an 4 hour per week student's course.

1. The company has to invest time (i.e. money), to look at libraries that might fit its needs.
2. The company has then to evaluate, i.e. take a closer look at, the libraries that made it through the previous step. If more then one library shall be employed, then interoparability needs to be tested also.
3. Once the presumably best library has been chosen, it needs to be employed in an actual commercial project. Evidently this is more costly than the last step, where usually only a single employee's time is invested, with no side effects on other parts of the company, whereas here a whole project might stall, if the library does not hold up to a real life situation.
4. Finally the company has to invest into bugfixing and extending the libraries functionality to better adapt it to its needs. Bugs that pop up in the library when a product has shipped are evidently a great financial risk[62]

8.9.3 Commercial vs Open Source

An interesting thing to note is, that much the same steps/costs occur for any kind of software technology/library, whether it is in the public domain, or under some sort of open source or commercial license.

A clear advantage of commercial libraries here is, that as long as the company who sells the libraries stays in business, it will be responsible for fixing bugs and extending the functionality of its library according to its customers' needs, simply to stay in business.

In addition, serious bugs in a commercial library can seriously damage the companies reputation, again hurting business, so there is an incentive to avoid such bugs.[63]

Another important aspect in the evaluation of software libraries by a company are Intellectual Property "**IP**" infringement issues[64]. License agreements in commercial libraries usually do legally safeguard the buyer against all or at least most of these infringement issues.

[62] More or less so, depending on the specific business field; i.e. a serious bug in a console video game is very bad, whereas in a PC game customers are more used to using bugfix patches; on consoles it might not even technically possible to patch the executable to fix the bug, depending on the console type and whether it is connected to the internet.

[63] Note that due to the costs and difficulties of switching a library in use, this might not be as harmful to a company selling software libraries as it sounds, as long as bugs are fixed in a timely manner.

[64] copyright infringement, patent infringement,...

Public domain and open source licenses usually have no such safeguard clauses. This is evidently quite natural, since the developers are not making any money with the libraries, so there is not even a possibility for them to take over the costs of legal disputes.[65]

Big open source projects, which have existed for some time, have a relatively stable contributor base and especially a large user base, which includes commercial use, have, in a sense, a natural "immunization" against IP infringement: The stable contributor base is usually close nit and takes the project very seriously, so none of these people will introduce code that is e.g. taken from another library or a commercial code base. In addition, since the project has existed for some time, any IP infringement in at least the core code base has a large likelihood of having been exposed already. For software patent infringements the situation is, as ever, the most volatile, due to the large number of trivial patents that have been issued, especially in the US. But also in this case, anybody using the library will, due to the large user base, at least not be alone, in case of an alleged infringement; the number of users gives strength, especially if IT heavyweights with an open source agenda, such as IBM, are employing the library in their products.

On the other hand, public domain and open source software has the advantage that you get unlimited access to the sourcecode[66] and you can download[67] and start evaluating it immediately.

Additionally the programmers behind an open source project are usually much more accessible, and, at least for large, well known projects the large user base is usually a good source for support[68]. In addition, a large user base can in itself be seen as at least a strong indication that the library is stable and useable.

[65] In the author's impression, in practice this thought is not even entertained, stopping at a (understandable) "Hey, you are getting it for free. What do you want ?" attitude

[66] This option is available with many commercial libraries, but usually at greatly increased extra cost. If a commercial evaluation license is not a (full) sourcecode license, then the evaluation will be less accurate, since the reviewer will not be able to assess how well a library's sourcecode is written and documented.

[67] copy it from a DVD,...

[68] albeit with no guaranteed reaction time and at times not completely reliable; it might also be more difficult to get "support" for more complicated questions. In addition, posting questions about a sensitive part of a commercial software project into a public-access forum might be problematic.

8.9.4 Graphics Libraries

The author would like to point out that graphics libraries are a relatively problematic type of library, since what most companies require is a full game/3D middleware solution, which supplies scene management & loading, basic rendering, lighting, shadows and effects rendering, animations, collision detection / physics simulation, saving of the world state, sound, user input,... For larger companies all of this combined with multi-platform support for PC and as many video game platforms[69] as possible.

Since the GTP supplies only a small part of this, which is supposed to sit on top of it[70], it faces the additional problem of needing to be compatible or at least easily integrable with the existing "base" library.

8.9.5 The GameTools Libraries

So what does that all mean for the "free" libraries created within the GameTools Project ?

Unfortunately, the GTP is somewhere in the middle, combining some of the worse properties of both public domain/open source and commercial software libraries: Potential users cannot immediately download and evaluate the code and their is also no large user base. Documentation of the libraries can also not be evaluated before signing the GTP SIG membership agreement.[71] On the other hand, he also does not get any kind of security against IP infringement[72]; in addition to not having a proven code base and large user base, this is more problematic in the case of the GTP, since potential SIG members were of course aware that the GTP libraries were created at universities, an environment not known for its strict adherence to any kind of license or IP rules.

The next section details the effort necessary to become a SIG member.

[69] Playstation 1, 2 & 3; Xbox, Xbox 360, Gamecube, Wii,...; in some cases also mobile gaming platforms such as the Nintendo GameBoy series or Nokia N-Gage.

[70] a bit like a high tech cherry.

[71] see Section 8.5.2 for the reasons why the GTP offered no evaluation license.

[72] The author lobbied for supplying at least a limited IP infringement security, e.g. excluding patent infringement; this proposal was turned down, advocated by the GTP PC. While the author can understand that the participating universities (industrial consortium members did not supply code to the GTP library) did not relish the thought of guaranteeing that no one of their researchers/developers had copy & pasted code from an intellectually protected source into the GTP libraries, it unfortunately does not build trust with companies considering to become SIG members; especially larger companies are naturally much more sensitive to these issues, because they are much wealthier targets for IP infringement lawsuits.

Becoming a GTP SIG Member

To get access to GTP technology, a prospective SIG member first has to invest time to scour the available information, i.e. the information on the GTP webpage, especially the videos/demos or information that e.g. GTP CM supplies.

He then needs to contact GTP CM, to inquire if anything would prohibit him from becoming a GTP SIG member.[73]

After this has convinced him to proceed, he needs then to download the GTP SIG agreement, and either read and evaluate its implications himself, or pass it on to a lawyer. Even though the agreement was written to be as short, concise and lenient as possible, it is still a legal document, and therefore needs some time to assess; in addition such decisions usually have to go through the management of a company, since a developer cannot pass it to a lawyer or the legal department on his own.

After the agreement was greenlighted, it has to be filled out, someone with the authority to do so has to sign it on every page, and it then needs to be faxed to two fax numbers, at VUT, FAO[74] GTP CM, and at GTP HQ at UdG, FAO GTP PM[75]

One can see that the amount of work (and therefore money) and trust a potential SIG member has to put in up to this point, where he becomes a SIG member, is considerable. Even at this point, it is clear, that the GTP libraries are far from being free in the sense that they incur no costs.

Evaluating the GTP Libraries

After having emailed GTP CM that the fax had been sent, the newly made SIG member would receive instructions on how to create an account int the GTP support forum (see Section 8.6), through which he would get his access data to the GTP software repository (see Section 8.5.5) through a private forum message.[76]

After he had got the software repository access information (and having downloaded and installed a Subversion client, if need be), he can check out[77] the

[73] The author got e.g. inquiries from private persons, universities and companies outside of the EU, asking if they could get access to GTP technology.

[74] abbr.: for the attention of

[75] Initially it was faxed only to GTP CM to keep the process simpler, but GTP PC requested that it also be faxed to Girona.

[76] this was done, so that new SIG members were aware that the forum existed and definitely had a valid account, in addition to avoiding sending the access data via email.

[77] "download"

several GB that are currently stored in the GTP software repository, and can then finally start to evaluate the libraries.

8.9.6 Future Costs

All of the costs (effort) described above, is in addition to the costs a potential SIG member can anticipate with regards to missing or bad documentation, support, bugfixing and costs of integrating the libraries into his own framework. Even if documentation and support turn out to be excellent, bugfixing fast and the integration costs low, then the GTP still has a known limited lifespan[78], and support and bugfixing are therefore unclear after that.

8.9.7 Conclusion: GTP Technology is Not Free

From all of the above one can clearly see that GTP technology is not free for GTP SIG members, whether they actually chose to use it or not.

Even in the sense that “no money is transferred from the SIG member to the GTP or its partners” the GTP libraries are finally not free, due to the GTP EU contract which calls for charging SIG members for support and future extensions after the end of the project.

8.10 Taking Stock: An Early Postmortem

[78] contrary to a company or an open source project

Figure 8.7: Current look of the GTP start page at `http://www.gametools.org/`.

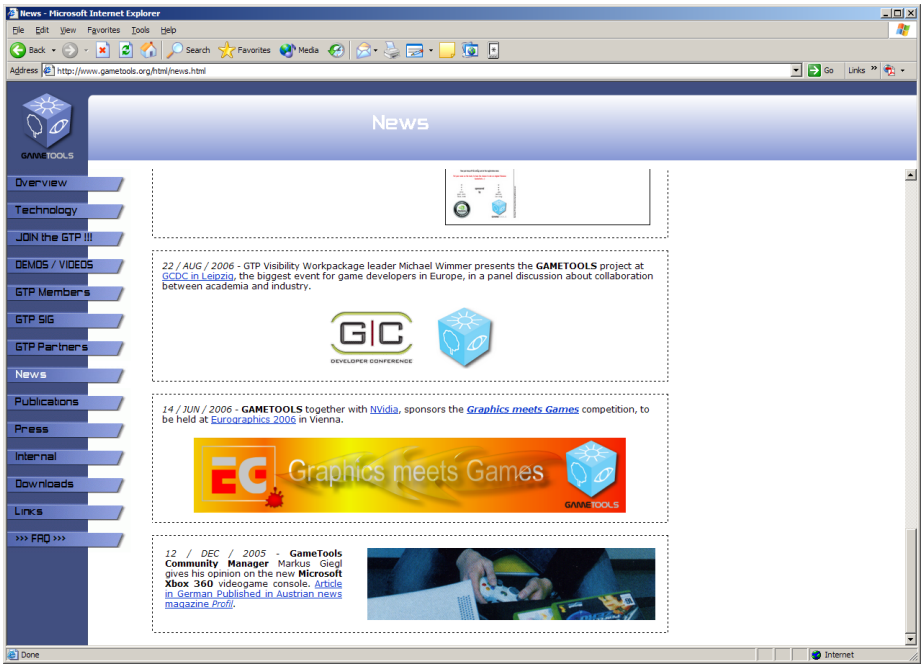


Figure 8.8: GTP webpage News section
(`http://www.gametools.org/html/news.html`)

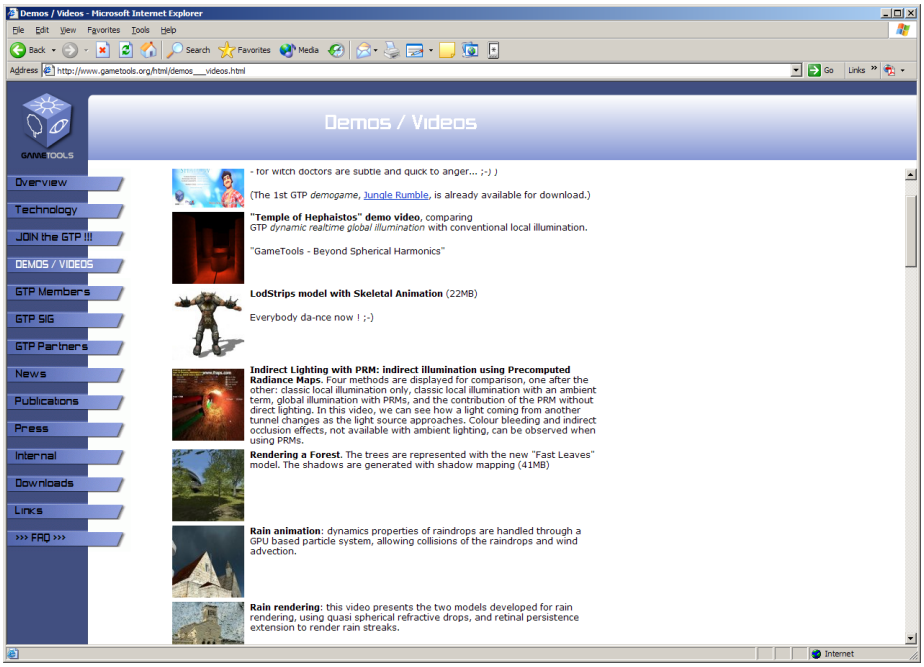


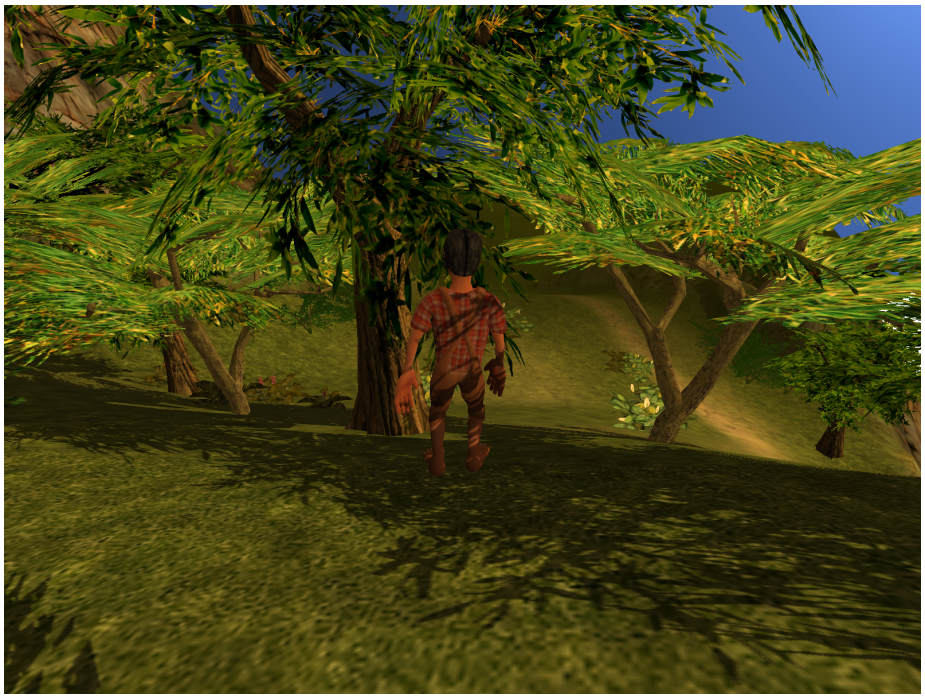
Figure 8.9: GTP webpage Videos/Demos section
(`http://www.gametools.org/html/demos___videos.html`)

Figure 8.10: GameTools advert appearing in April 2007 edition of Game Developer magazine (`http://www.gdmag.com/`).

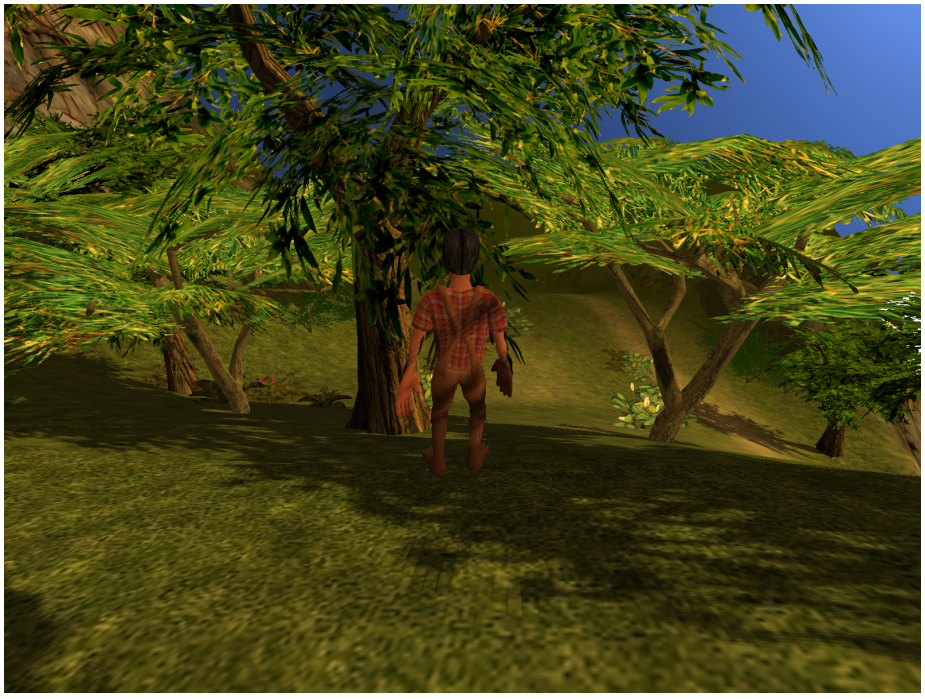
(a) conventional decal imposters

(b) GTP depth imposters

Figure 8.11: Screenshots from the volcano level of GTP demogame Jungle Rumble, comparing conventional, flat decal imposters with GTP depth imposters.

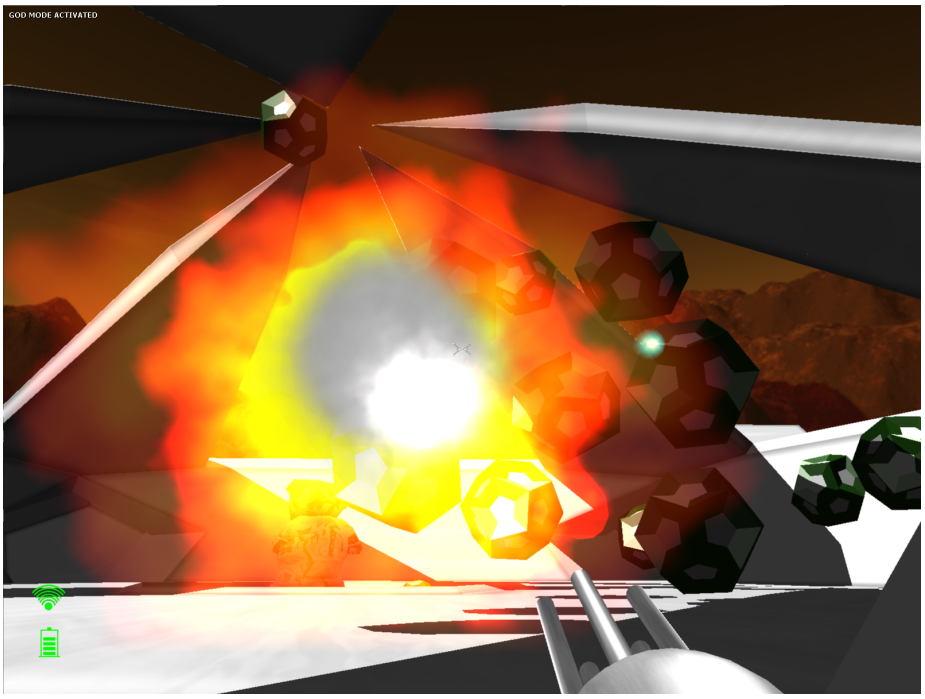


(a) LiSPSM (optimal case)

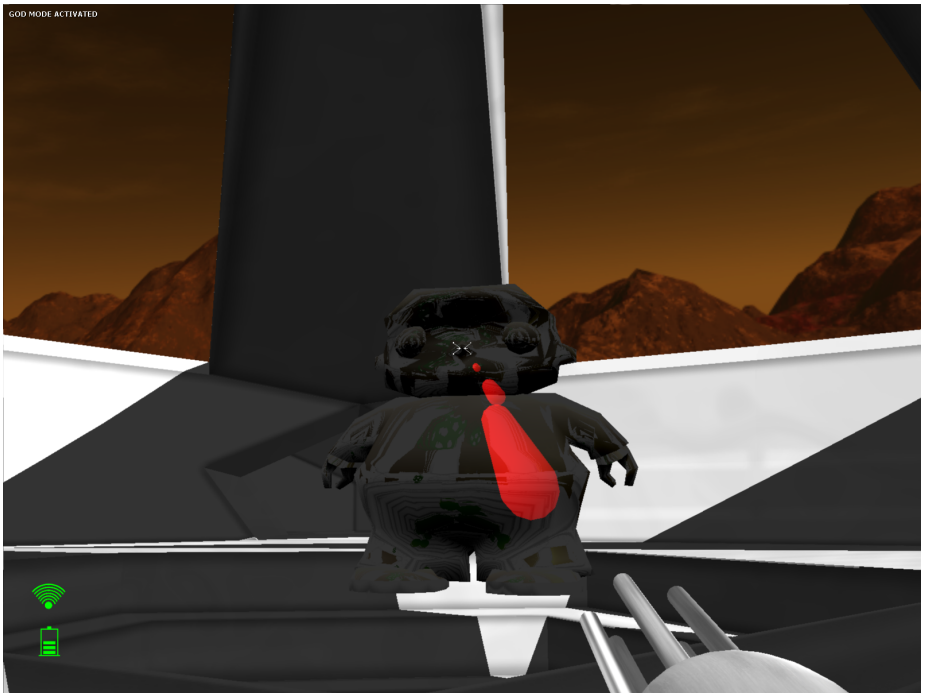


(b) uniform SMing

Figure 8.12: Screenshots from the GTP demogame Have U Seen My Shadow. LiSPSM shadows are compared with uniform shadow mapping. Vegetation and terrain use the author's Unpopping algorithm.



(a) Penta-G rotating blaster gun in action



(b) GTP Raytrace Effects on a weird looking statue

Figure 8.13: Screenshots from GTP demogame Penta-G; the original Penta-G concept by the author (shown here) explored a world design based on a 5-sided symmetry. This has unfortunately been watered down to a generically looking commercial FPS rip-off look in newer versions.

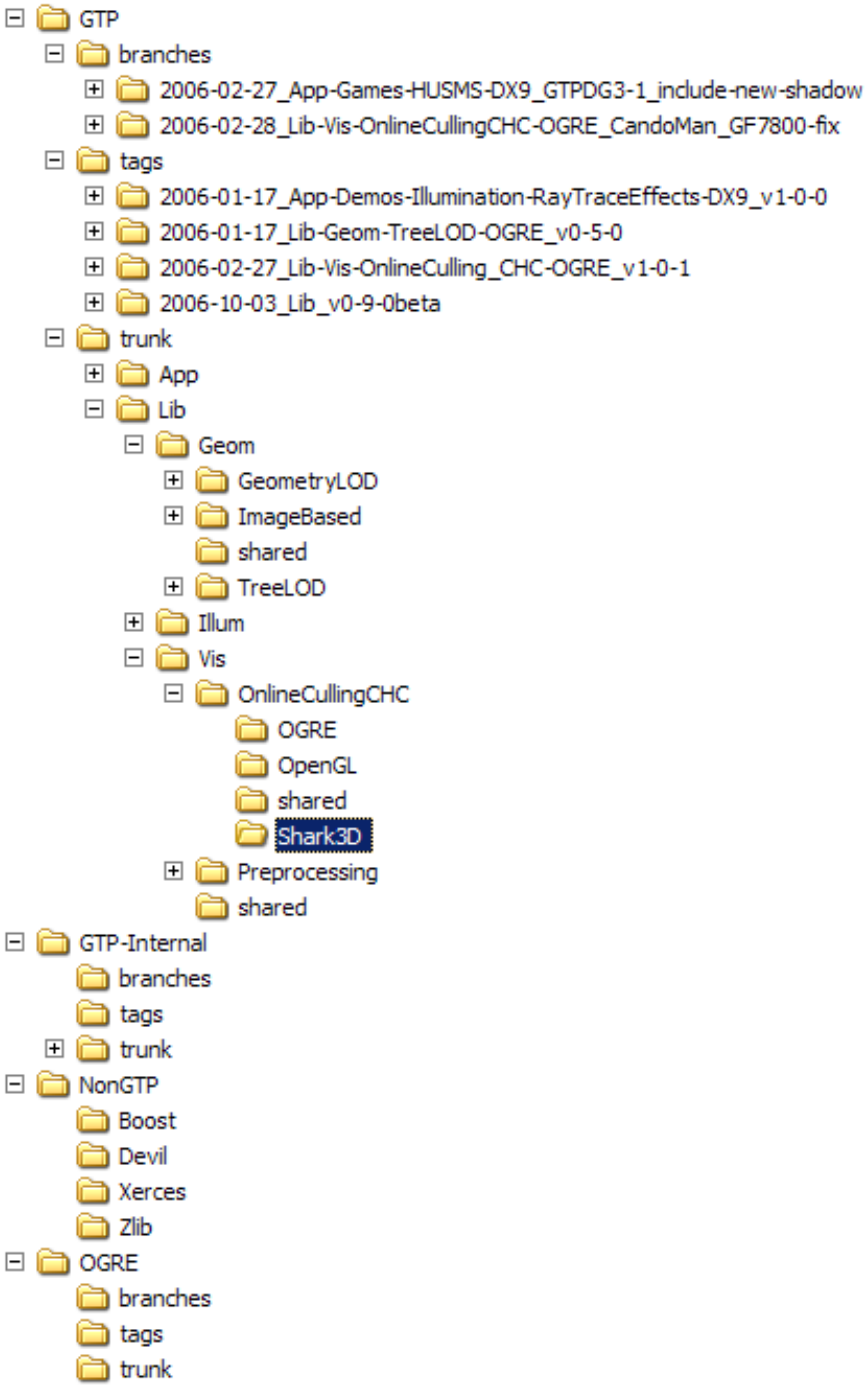
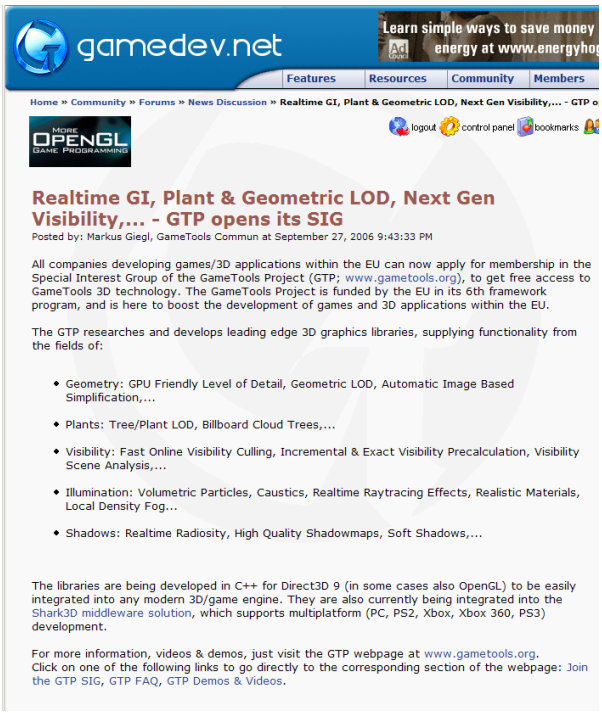


Figure 8.14: The 1.0 version of the GTP software repository scheme.

Figure 8.15: The GTP support forum start page at `http://forums.gametools.org/`.

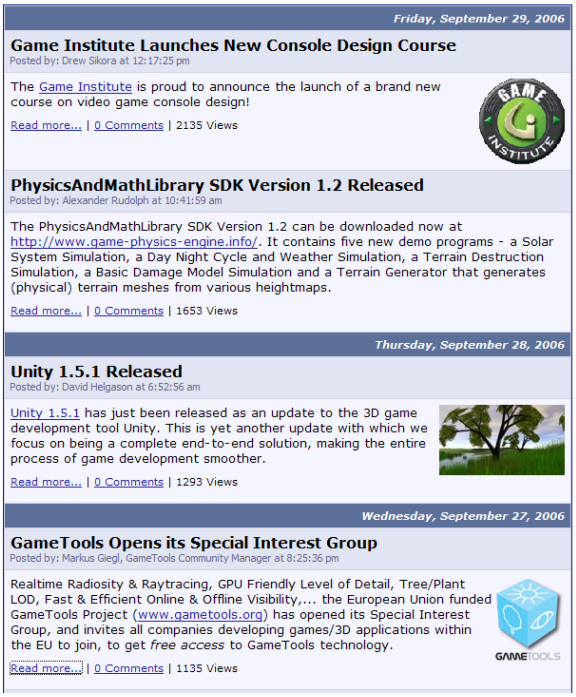
(a) gamasutra.com



(b) gamedev.net

Figure 8.16: Examples of free GTP advertising on the web on several important game development news/discussion sites.

(c) gamedev.org



(d) devmaster.net

Figure 8.16: (continued) Examples of free GTP advertising on the web on several important game development news/discussion sites.

Figure 8.17: GameTools exposure in the December 2005 issue of renowned Austrian news magazine “profil”.

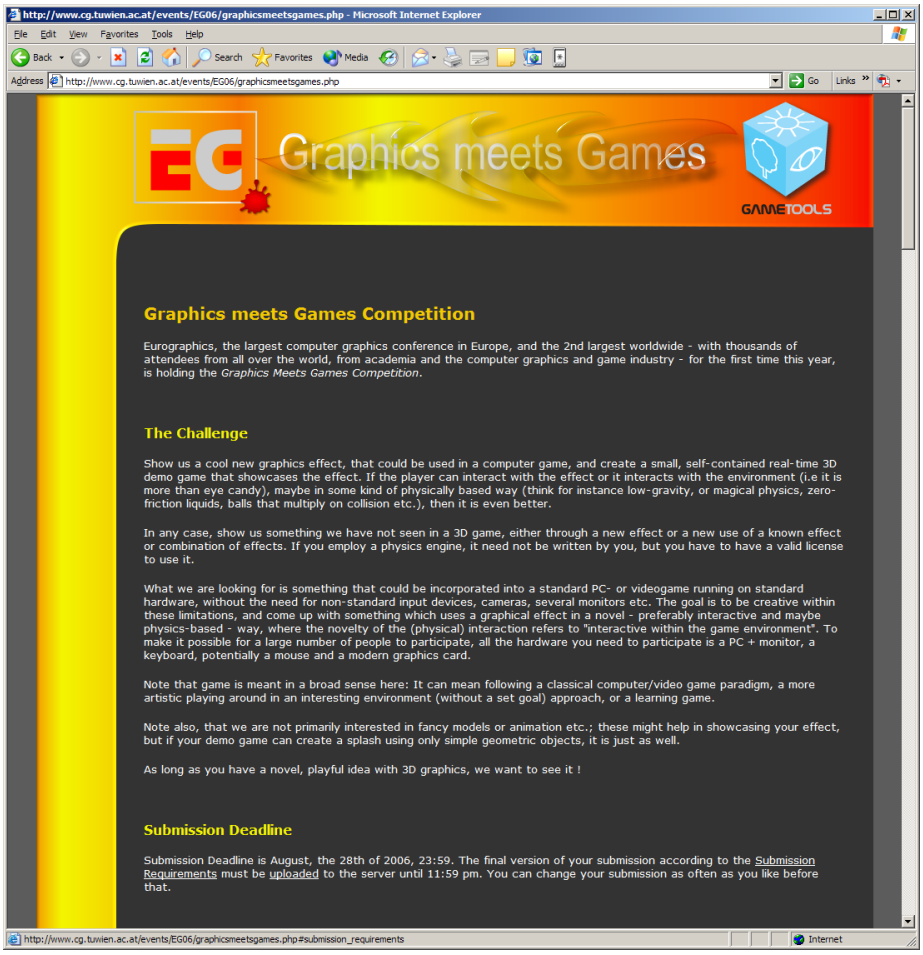


Figure 8.18: GTP at the Graphics meets Games competition at Eurographics 2006: GmG webpage.

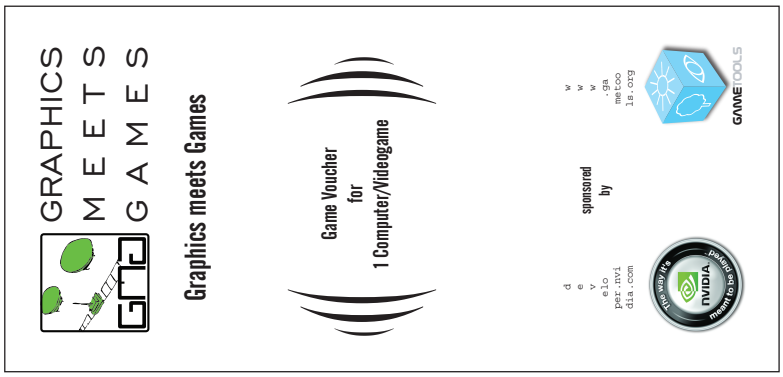


Figure 8.19: GTP at the Graphics meets Games competition at Eurographics 2006: Game prize voucher.

Figure 8.20: GTP poster for Resfest 2006.

GameTools Project
Special Interest Group Membership Agreement

between

The GameTools Project
University of Girona
Plaça Sant Domènec, 3, 17071, Girona
Spain

and

Company	
Address	
Telephone	
FAX	
Webpage	

represented by

Name (First Last)	
Position / Job Title	
Telephone	
Mobile	
FAX	
email	

By signing this agreement, the undersigned also states that he/she is authorized to enter into this legally binding agreement on behalf of his company. To prove the authenticity of the content of each page of this agreement, each page needs to be signed.

(Place, Date)

(Signature)

Figure 8.21: The GTP SIG Membership Agreement (4 pages).

Sense and Purpose of the Agreement

The GameTools Project (from here on "GTP", http://www.gametools.org/) develops next generation 3D libraries and is funded by the European Union in its 6th framework programme. This agreement has the main purpose of allowing members of the GTP Special Interest Group (from here on "SIG") to use the technology developed in the GTP as a compiled part of their products at no charge to strengthen the EU economy/job market, while at the same time strengthening the GTP consortium members by giving them exclusive rights to publish scientific results about GTP research results and sell it as a software development library.

General Part

Copyright & Ownership

The Software Package (programs, libraries, sourcecode and documentation) is owned by the respective GTP core consortium members.
All rights reserved.

Distribution

You must not make any part of the software package accessible to a third party.

Confidentiality

All information included in the software package is strictly confidential and cannot be communicated to any third party, unless you got this information lawfully from another source.

Transferability

The rights you acquire through this agreement are not transferable to any third party.

Termination

Should you violate any of its terms, you agree that the GTP can at any time terminate this agreement and all the rights you have obtained through it. In that case you agree to render all copies of the software package and any software based on it immediately unusable.

(Place, Date)

(Signature)

Disclaimer

You are aware and agree that you use the Software Package at your sole risk and that it is provided to you "as is" (i.e. without warranty of any kind, express, statutory, implied or otherwise, including without limitation any warranty of merchantability or fitness for a particular purpose). No warranty for the software will be created by any information or advice you are given. You also may not rely on any such information or advice.

Liability

Under no circumstances shall anyone participating in any way in the GTP be liable to you for any special, consequential, incidental or indirect damages of any kind (including without limitation the cost of cover, damages arising from use, profits, loss of data, goodwill, or property damage), whether or not he/she has been advised of the possibility of such loss, however caused and on any theory of liability arising out of this agreement. These limitations shall apply notwithstanding the failure of essential purpose of any limited remedy.

Governing Law and Jurisdiction

This Agreement is governed by and construed and enforced in accordance with the internal laws of Spain, and shall be binding upon the parties hereto worldwide. The exclusive jurisdiction to adjudicate any dispute arising out of this Agreement shall lie with the courts within Barcelona. Barcelona, Spain is the place of jurisdiction.

Severability

The validity of the remaining terms and conditions shall not be affected, if any provision of this agreement becomes ineffective or void. In that case the contracting parties shall replace the void or ineffective stipulation with another effective regulation appropriate to the sense and purpose of the agreement.

Specific Part

SIG Membership Official

When becoming a SIG member you agree for this to be official.
You agree specifically that the GTP may use your name and/or logo to publish the fact that you are a SIG member.
This includes but is not limited to that you agree to your name and/or logo being listed on the GTP webpage, being included in a press release, talk, video, tv-show or any other publication about the GTP.

_____
(Place, Date)

_____
(Signature)

The use of your name & logo is, in any case, limited to expressing solely your SIG membership, and will not be used for any other purpose by the GTP without your prior consent.

Use of GTP technology/software

By signing this contract and thereby becoming a SIG member you will get preliminary access to the technology/software as it is being developed by the GTP, and you are allowed to use the technology/software for any purpose, including commercial purposes, condition to the following restrictions:

EU Focus

All of the software development using the Software Package has to be done in the EU. If the SIG member is an international company which also has development studios outside of the EU and this condition cannot therefore be fulfilled for practical reasons, then most of the Software development using the Software Package has to be be done in the EU.

Credit

If you make a work/program that is based in whole or part on the Software Package available to a third party you have to give clear credit to the GTP, by including the GTP logo with the text "based on technology from the GameTools Project (www.gametools.org)" below it in either your work's/program's manual/documentation and/or credits.

Scientific Results

You are not allowed to publish (including but not limited to at conferences, in journals or books and over the web) research results, based on and about technology developed in the GTP during the duration of the GameTools Project.

Library Creation

You are not allowed to make libraries (either compiled or in sourcecode form), based on the Software Package or parts of it, available to third parties in any form which can be used for software development.

(Place, Date)

(Signature)

Chapter 9

Game Development and the Game Industry

The following short chapters present thoughts and observations by the author about different topics in game development and the game industry. They address questions and topics the author has either come across repeatedly with regards to game development[1] (“Why Buying a Computer Game is Not Like Going to the Movies”, “Why Computer Games are Seldom Art”), or which the author has developed himself and thinks could help in not wasting resources in game development (“The Quality Frontier in 3D Games”).

[1] e.g. in Game Developer magazine

Chapter 10

Why Buying a Computer Game is Not Like Going to the Movies

The one industry the game industry is most often compared with is the movie industry. And there are certainly a lot of parallels: Both are creative industries, both are done by large teams of people working together (as compared to e.g. literature, which is usually done by one individual alone), both create a comparable amount of revenue and both are “the winner takes it all” economic areas, where only a small percentage of top titles create nearly all the profit (and are desperately needed to offset the losses of the rest). But there is one major difference, namely that playing computer and video games does not not nearly have the same mass appeal as going to the movies. Apart from very few exceptions (e.g. ”The Sims”), computer and video games are mostly played by the male half of the population. And within this male half, it is still a specific subgroup that is not nearly as large as the number of the male population that goes to see movies. This chapter deals with some observations concerning differences between the two industries, and concludes with what could be done to improve the situation.

10.1 Recognizing Creative Minds

Recognition of the creative mind(s) behind a computer game is the exception, not the norm (Peter Molyneux, Sid Meier, false example: ”John Carmack”),

contrary to movies, where at least directors are stars in their own right, and are recognized by the audience. Movie directors (and to a lesser extent producers) are brands: If they make a bad movie (pick the wrong script, wrong actors,...), their reputation is diminished, not the one of the film studio. Computer games on the contrary are created by ”faceless drones” as far as the gamer is concerned, and are usually mostly associated with the publisher. What matters though, are the actual people creating a movie, or a computer/video game.

10.2 Communication

Seeing a movie allows one to talk about it with your partner, friends & coworkers. That this is possible because of two reasons: First going to the movies is still a leisure activity that many more people participate in (the games industry might have overtaken Hollywood in terms of money [because games are so extremely expensive], but certainly not in terms of people worldwide participating in playing them), and second the likelihood that many of the people you know have seen the same movies as you did is much higher than that them having played the same games (if they play games at all): Going to the movies is a short event as compared to playing a computer game, while at the same time being much cheaper (even assuming that you pay for 2 tickets, it should cost no more than 16 Euro to see a movie, as compared to 50+ Euro to buy one computer game), so many people can watch several movies per months.

10.3 Pricing

The longevity of computer games combined with their high price poses a problem of its own kind: If a computer game only offers a short playing experience, then players will be even less inclined to buy it (or its sequel); but since the playing experience is much longer than a movie, there is room for less games in peoples lives. Therefore games such as the hugely successful GTA series can pose a huge problem for other publishers, since the games, being released in a regular fashion, occupy so much of gamers time and money. This problem is amplified in massive multiplayer games, where players will normally be subscribed to one game at a time, playing them for periods of several months at least.

10.4 Social Activity

Going to the movies is (currently) a much more social activity than playing a computer game: Most people do it together with their partner and/or friends, it is part of a whole bunch of evening activities, so even if the movie turns out to be only mildly entertaining, the overall experience can still be fun.

10.5 Ease of Picking a Good One

Picking a movie which you will at least enjoy a little bit is much easier than doing the same thing with a computer game; Computer games have so much more elements that one could not like (e.g. for me control mechanism of most 3rd person videogames, such as ”The Simpsons”, ”Rayman 3D” etc) that the likelihood of picking the right one is much smaller.

10.6 Fun at Picking a Bad One

Even if the movie was really bad, although no one would deliberately go to see it (apart from Star Wars Episode 3, maybe), one can still have fun taking it apart with other people who didn't like it. With a computer game, if it turns out to be bad, that's it: You spent a lot of money, and all you got in return is this colorful box, that shouts at you how good the game supposedly is, and which you won't throw away, because it cost a lot of money, only to have it remind you how much the game that came in the box sucked, every time you look at it.

10.7 Time Investment

If a movie has its weak sections, then in most cases the viewer will still watch the whole movie (because it lasts less than 2 hours or the people he went to the movie with don't want to leave or...), and therefore might not be (as) disappointed by the whole experience. Playing a computer game takes a lot of time investment on the part of the player, and if a game is unenjoyable over a longer stretch of time, then the player will most likely stop playing it, and therefore never get to the more enjoyable sections; this is most pronounced at the beginning of a game, where the player has to learn the controls and the subtle differences in game logic that even games of the same genre exhibit

(e.g. ”do I have to check every door if it can be opened, or are ther visual/sound clues telling me, which ones are just dummies”). Actually this initially happened to me with one of the best games I have ever played, Deus Ex 1: I downloaded & played the demo, which consists of the first level of the game, and was very dissapointed: The graphics were not very good, there was not much of the - supposedly excellent - story to be seen and I wasn't too convinced about the basic FPS game mechanics. So I decided against buying it; fortunately I found it in a bargain bin some time later and decided to give it another try, mostly because it was the american version, and therefore came in a funny box, and it was really cheap - about the price of a movie ticket, if I recall correctly.

10.8 Language Barrier

Language of game: Free to choose (e.g. System Shock 1)

10.9 Controls

Controls of a game: No need to learn to ”control your movie experience”.

10.10 Bugfixes

No ”bugfixes” for movies.

10.11 Bootlegging

Bootleg Copies: Illegal copies are of course a problem for both the movie and the game industry, and at first glance it might seem as if both suffer the same from people using their product without paying for it. But is that really so ? Let's first observe that watching a bootlegged movie is, of course, like watching the DVD to a film, and not like going to the movies: The whole social experience of going out, meeting friends etc that has been discussed above is missing in addition to not being presented the movie on abig screen with THX sound. But even with the bootlegged copy of the movie (with DVD sales of course being a big source of revenue for the movie industry) the experience the viewer gets is normally not the same as with the original product: Either the bootleg

is filmed off a movie screen, which gives such horrendous quality, that one must deduce that these people really don't want to spend the money at all; or the DVD has been downsampled to fit on a 4.7GB write-once DVD (while this is in the process of changing, the advent of high definition movies will probably balance the scales in this area once more). On the other hand with computer games, when you use a bootleg copy, you get at least the same playing experience than with a legal copy; I say "at least", because ironically because the cracked copies have the copy protection mechanism removed, saving the player the hassle of having one of the games discs in his drive when starting the game. The only thing the player using the bootleg copy is missing is the box the game comes in (increasingly just a standard DVD box, so not much there) and the handbook, something which most games don't need nowadays, and which has been reduced to such an extent that it is pointless to read it anyway, if you have any experience playing games ("the left mouse button fires the weapon" - really ?). In addition again the high cost of computer games lures people to copy them: You can pick up a DVD for 5 Euros, freshly released they cost 15-20 Euros, which is still only around one-third of the price of a computer game.

10.12 How to Improve the Situation

Now that we have seen that there are in fact a lot of differences between the game and movie industry with regards to its customers, what could be done to improve the situation ? Buyers can win something (which money can't buy, e.g. a trip to a big developer studio, a trip to E3 with backstage pass, etc). Buyers can return the game for a refund (maybe in the form of a voucher). This could be extended as follows: By some technical mean (connection to a server on the internet, rub off card shipped within the gamebox etc) the player unlocks level after level of a game. If he stops playing the game, he is refunded the levels he did not play. Alternatively this could be turned around, so that buying a game is very cheap, and the player has to buy the levels before he plays them. Both approaches, of course, require the publisher to be quite sure that players will like their products from start to end.

Age: Movies have been around for much longer, so older people know them. But still: Many people feel that, once they reach a certain age, they are "too old to play electronic games".

Money: In poor countries people can go to the movies, but they cannot afford the expensive equipment necessary to play electronic games.

Chapter 11

The Quality Frontier in 3D Games

This part of the thesis introduces the concept of the “quality frontier”. In general the quality frontier is the curve through the quality of all the parts that make up a whole thing. In a 3D computer/video game it is the curve through the quality of all the areas that make up the game, such as graphics, sound, controls, artificial intelligence,.... A very important part of modern 3D game is graphics, and the quality frontier here again runs through the different subareas, such as water, light, shadows, surfaces,... rendering. Quality here is measured as compared to an ideal rendering. For example for most first person shooters the ”ideal rendering” would be to have everything look like in the real world, with maybe a touch of hyperrealism. Therefore each graphical area can be categorized as to how far it is from that goal. The thing one wants to avoid at all costs is, to have a certain area of the rendering shatter the illusion of the whole. Example: If a game world is depicted with the graphical quality of Doom 1, no one would expect to see puddles on the floor that reflect the environment distorted through physically correctly modeled ripples on its surface; would one include such an effect into the engine, one would not get better overall graphical quality, but worse, since the perfectly rendered puddle would only highlight how badly the rest of the environment is depicted in comparison ! Therefore the goal must be to have a quality frontier curve which is as uniform as possible. In practice that also means, that it does not make sense to invest money & effort to push the graphical quality in certain areas of a game only, while others lag behind. The goal has to be to advance the quality frontier as uniformly as possible. The principle of the quality frontier also holds for other areas of game

development: The more realistically computer controlled allies (henchmen) to the player are rendered, the more it becomes obvious if their animation (or animation blending) or AI cannot keep up. Good examples for a uniform quality frontier are ”Half-Life 2” and ”Unreal Tournament 2003/2004”. Half-Life 2 for instance is the first game that can hide the typical ”water collides hard with the shore because of depth buffer” problem by blending the critical area near the shore (which, on closer inspection is of course not a perfect solution, but which withstands all but the closest scrutiny, and will certainly not disrupt the visual quality of the game while playing it). ”Unreal Tournament 2003” introduced diagonal sidestep animations for opponents, making the combat sequences the player experiences much more believable (negative examples along that line include Quake 3 and games based on this engine; for instance in Jedi Knight 2, even the storm troopers backpedalled and strafed left/right while displaying a simple simply ”moving their legs up and down”-animation - in this regard even the stormtroopers from the original Star Wars first person shooter, Dark Forces 1, had looked better). Another negative example is Doom 3, which uses shadow volumes together with too small textures (it is amazing how much textures the game evidently uses, even though it looks so similar everywhere one goes), leading to ridiculously blurry surfaces from even a moderate distance which clash violently with the razor sharp shadow volume edges casting shadows onto the surfaces; in its predecessors only low resolution light maps where used (which are still present in Doom 3 and work fine there), avoiding such problems. The only exception to the ”uniform quality frontier”-rule is, if you are trying to sell your game on being the first to feature a specific graphical effect. In this case you most certainly will not increase the quality of your game, but due to a certain tech savvy buyers and a greater echo in the games press, you might shift more units.

Chapter 12

Why Computer Games are Seldom Art

1. Games ⇒ Playing ⇒ Entertainment
2. excludes most negative emotions, that touch us in a higher place, such as art does.
3. So we need another term in any case (”Virtual Interactive Art” ?)
4. Most people do not consume art in any form, why should they in games ?
5. Creating a computer game is the most time consuming way of expressing oneself, short of a Hollywood movie. So the question is, if it is a good medium for artistic expression ? What are its unique characteristica, which distinguish it from all other forms ?
6. INTERACTION (Chracteristicum #1): The ”game” reacts to another persons actions or it tries to coerce reactions to its own action (feedback); it can mix the real environment with the virtual gameworld (face textures, integrate real world geometry into game, etc)
7. Isn’t art mostly about silent reflection ? Or is this just cultural prjudice ?
8. Art in commercial games, problems:
   (a) Ever increrasingly expensive to make ⇒ must be ”surfire” hits. Hollywood does seldomly produce art either (but it does sometimes come near !).
   (b) Target audience is mostly 14 year old males. Probably the part of the population that has the least connection to art.

9. Time you have to invest into it as a ”player”: If you don’t like a painting, photograph, etc artwork, or it does not touch you, you can simply move on to the next. Not so in a game.
10. ”Playing games is something that children do”
11. Only artistic game: ”Deus Ex 1”
    (a) Not everything is as it seems
    (b) What is ”good” and what is ”evil” ?
    (c) What makes a human ”human” ? What can you take away from a human, without him losing his soul (Cyborgs ”monsters”) ?
    (d) Is it OK to manipulate the world ”for its own good” ?
    (e) End question: What future world will you create ? Reflect about utopian future. 3 choices:
        i. Take control over the world by merging with an AI, becoming a benevolent dictator
        ii. Plunge world into dark age (technological breakdown)
        iii. Leave everything as it is (Illuminati fractions scheming in the background and controlling the wold through manipulation)

Choice one seemed tempting: You could right all wrong, bring peace to the planet, etc. But: Benevolent dictator is a scheme that has gone wrong time and time again. What if the AI is the dominant part ? What if you loose your humanity, immediately or over time ? What if you go mad immediately or after some time ? What if absolute power corrupts you absolute ? Or even in the best case: Are you really sure, you know what is best for everyone ? Suddenly it did not seem to be such a good choice anymore. Choice two was the next favorite: Break the manipulative grip of the Illuminati over the world, destroy the evil technology that allowed for mass manipulation and control. Give people back their freedom ! But what would such a world be like ? A new worldwide middle age, where the physically strongest dominate the weaker ones; where local warlords and kings take what they like and do how they please (”ius prime nocte”). Where control over remnants of technology gives great power. And a world that in the end, would probably even end up where it is right now. So to my own surprise I picked option three, seeing the relative freedom of the people to do what they want and aspire for what they choose, a high standard of living, free choice of religion etc as the best of three bad choices: A world, even if manipulated by warring factions

of humans at the highest level seemed better than one dominated by one not-quite-human dictator or one where the strongest, most brutal individuals where the most likely to wield absolute power in local areas. Of course, given the choice I would have opted for choice 4: Eliminate the Illuminati, and bring the world one step closer to being a fair, democratic, utopian society. But be wary: Just as when you elimite all bacteria and fungi on an old painting, all you achieve might be to only to replace it with the fastest growing one, which does more damage to the painting than the power balanced heterogenous scenario. Why where these questions so important to me in the first place ? The answer is: Because I had become emotionally involved with the world that Deus Ex portrays, by fighting in it, and fighting to reach this decision point (although, of course, expecting that the decision would have been made for me by the game designer). So the interactivity, which we described as one of the main distinguishing factors of electronic games, plays a vital role in making the decision emotional and engaging.

This is where a commercial game can give an experience which might, in a sense, make it valid to call it a work of art.

Bibliography

[AL04] Timo Aila and Samuli Laine. Alias-free shadow maps. In *Proc. Eurographics Symposium on Rendering 2004*, pages 161–166, 2004. 19

[Arv04] Jukka Arvo. Tiled shadow maps. In *Proc. Computer Graphics International 2004*, pages 240–247, 2004. 21

[AWG78] Peter Atherton, Kevin Weiler, and Donald Greenberg. Polygon shadow generation. In *SIGGRAPH '78: Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, pages 275–281, New York, NY, USA, 1978. ACM Press. 8

[BAS02] Stefan Brabec, Thomas Annen, and Hans-Peter Seidel. Practical shadow mapping. *Journal of Graphics Tools*, 7(4):9–18, 2002. 13, 18, 21

[Bli88] J. Blinn. Me and my (fake) shadow. *IEEE Comput. Graph. Appl.*, 8(1):82–86, 1988. 6

[BWPP04] Jiří Bittner, Michael Wimmer, Harald Piringer, and Werner Purgathofer. Coherent hierarchical culling: Hardware occlusion queries made useful. *Computer Graphics Forum (Proc. Eurographics 2004)*, 23(3):615–624, 2004. 25, 26, 123

[CD04] Eric Chan and Frédo Durand. An efficient hybrid shadow rendering algorithm. In *Proceedings of the Eurographics Symposium on Rendering*, pages 185–195. Eurographics Association, 2004. 22

[CG04] H. Chong and S. J. Gortler. A lixel for every pixel. In *Proceedings of Eurographics Symposium on Rendering 2004*, 2004. 21

[CGG+03] Paolo Cignoni, Fabio Ganovelli, Enrico Gobbetti, Fabio Marton, Federico Ponchio, and Roberto Scopigno. BDAM – batched dy-

namic adaptive meshes for high performance terrain visualization. *Computer Graphics Forum*, 22(3):505–514, 2003. 78

[CotEC04] Information Society Directorate-General Commission of the European Communities. Gametools - advanced tools for developing highly realistic computer games. *IST 6th Framework Programme, Contract Number 004363*, 2004. 96, 98, 99

[Cro77] Franklin C. Crow. Shadow algorithms for computer graphics. *Computer Graphics (Proc. ACM SIGGRAPH 77)*, 11(2):242–248, 1977. 11

[DL06] William Donnelly and Andrew Lauritzen. Variance shadow maps. In *SI3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 161–165, New York, NY, USA, 2006. ACM Press. 21

[EJ00] George Eckel and Ken Jones. *OpenGL Performer Programmer's Guide*. SGI techpubs library, 2000. Document Number 007-1680-060. 77

[Eng07] Wolfgang Engel. Cascaded shadow maps. In *ShaderX 5 - Advanced Rendering Techniques*, pages 197–206. Charles River Media, January 2007. ISBN 1-584-50499-4. 21

[FFBG01] Randima Fernando, Sebastian Fernandez, Kavita Bala, and Donald P. Greenberg. Adaptive shadow maps. In *Proc. ACM SIGGRAPH 2001*, pages 387–390, 2001. 21

[FS93] Thomas A. Funkhouser and Carlo H. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *SIGGRAPH 93 Conference Proceedings*, pages 247–254, 1993. 75

[GLY+03] Naga K. Govindaraju, Brandon Lloyd, Sung-Eui Yoon, Avneesh Sud, and Dinesh Manocha. Interactive shadow generation in complex environments. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 501–510, New York, NY, USA, 2003. ACM Press. 22

[GTGB84] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. Modeling the interaction of light between diffuse surfaces. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 213–222, New York, NY, USA, 1984. ACM Press. 5, 11

[GW07a] Markus Giegl and Michael Wimmer. Fitted virtual shadow maps. In *Graphics Interface Proceedings 2007*. CIPS, Canadian Human-Computer Communication Society, A K Peters, May 2007. 2

[GW07b] Markus Giegl and Michael Wimmer. Queried virtual shadow maps. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*. ACM Press, April 2007. 2

[GW07c] Markus Giegl and Michael Wimmer. Queried virtual shadow maps. In *ShaderX 5 - Advanced Rendering Techniques*, pages 249–262. Charles River Media, January 2007. ISBN 1-584-50499-4. 2, 42

[GW07d] Markus Giegl and Michael Wimmer. Unpopping: Solving the image-space blend problem for smooth discrete lod transition. *Computer Graphics Forum*, 26(1):46–49, March 2007. 2

[HLHS03] Jean-Marc Hasenfratz, Marc Lapierre, Nicolas Holzschuch, and François Sillion. A survey of real-time soft shadows algorithms. In *Eurographics State-of-the-Art Reports*, 2003. 17

[Hop96] Hugues Hoppe. Progressive meshes. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 99–108. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996. 78

[Hop98] Hugues Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *Proceedings of the conference on Visualization '98*, pages 35–42, 1998. 78

[JLBM05] Gregory S. Johnson, Juhyun Lee, Christopher A. Burns, and William R. Mark. The irregular z-buffer: Hardware acceleration for irregular data structures. *ACM Trans. Graph.*, 24(4):1462–1482, 2005. 19

[Kaj86] James T. Kajiya. The rendering equation. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 143–150, New York, NY, USA, 1986. ACM Press. 4, 5

[Koz04] Simon Kozlov. Perspective shadow maps: Care and feeding. *GPU Gems*, 1:214–244, 2004. 21

[KSS02] Jan Kautz, Peter-Pike Sloan, and John Snyder. Fast, arbitrary brdf shading for low-frequency lighting using spherical harmon-

ics. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, pages 291–296, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association. 14

[Lan02] H. Landis. Production-ready global illumination. *ACM SIGGRAPH 2002 Course Notes #16*, pages 87–101, 2002. 11

[LRC+02] David Luebke, Martin Reddy, Jonathan D. Cohen, Amitabh Varshney, Benjamin Watson, and Robert Huebner. *Level of Detail for 3D Graphics*. Elsevier Science Inc., 2002. see also www.lodbook.com. 75

[LSK+05] Aaron Lefohn, Shubhabrata Sengupta, Joe M. Kniss, Robert Strzodka, and John D. Owens. Dynamic adaptive shadow maps on graphics hardware. In *ACM SIGGRAPH 2005 Conference Abstracts and Applications*, 2005. 21

[LTYM06] Brandon Lloyd, David Tuft, Sung-eui Yoon, and Dinesh Manocha. Warping and partitioning for low error shadow maps. In *Proceedings of the Eurographics Symposium on Rendering 2006*, pages 215–226. Eurographics Association, 2006. 21

[McC00] Michael D. McCool. Shadow volume reconstruction from depth maps. *ACM Trans. Graph.*, 19(1):1–26, 2000. 22

[MH02] Tomas Möller and Eric Haines. *Real-Time Rendering, Second Edition*. A. K. Peters Limited, 2002. 2, 7, 12, 17

[MosBC] Moses. *Genesis*. approx. 1500-1200 BC. 1

[MT04] T. Martin and T.-S. Tan. Anti-aliasing and continuity with trapezoidal shadow maps. In *Proc. Eurographics Symposium on Rendering 2004*, pages 153–160, 2004. 21

[OAS+11] John Overall, Lancelot Andrewes, Hadrian Saravia, Richard Clarke, John Layfield, Robert Tighe, Francis Burleigh, Geoffrey King, Richard Thomson, and William Bedwell. *The Bible: King James Version*. Robert Barker, 1611. 1

[PC97] Stephen Prickett and Robert Carroll. *The Bible: Authorized King James Version (Oxford World's Classics)*. Oxford Paperbacks, 1997. 1

[RSC87] William T. Reeves, David H. Salesin, and Robert L. Cook. Rendering antialiased shadows with depth maps. *Computer Graphics (Proc. ACM SIGGRAPH 87)*, 21(4):283–291, 1987. 20

[SCH03] Pradeep Sen, Mike Cammarano, and Pat Hanrahan. Shadow silhouette maps. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 521–526, New York, NY, USA, 2003. ACM Press. 22

[Sch05] Daniel Scherzer. Shadow mapping of large environments. Master's thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, 8 2005. 17

[SD02] Marc Stamminger and George Drettakis. Perspective shadow maps. *ACM Transactions on Graphics*, 21(3):557–562, 2002. 21

[SK98] Laszlo Szirmay-Kalos. Stochastic methods in global illumination - state of the art report. Technical Report TR-186-2-98-23, Vienna, Austria, 1998. 5

[SKvW+92a] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul Haeberli. Fast shadows and lighting effects using texture mapping. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 249–252, New York, NY, USA, 1992. ACM Press. 9

[SKvW+92b] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul Haeberli. Fast shadows and lighting effects using texture mapping. *SIGGRAPH Comput. Graph.*, 26(2):249–252, 1992. 20

[SSKS06] László Szécsi, László Szirmay-Kalos, and Mateu Sbert. Light animation with precomputed light paths on the gpu. In *GI '06: Proceedings of the 2006 conference on Graphics interface*, pages 187–194, Toronto, Ont., Canada, Canada, 2006. Canadian Information Processing Society. 16

[Ste06] Christian Steiner. Shadow volumes in complex scenes. Master's thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, May 2006. 17

[TQJN99] Katsumi Tadamura, Xueying Qin, Guofang Jiao, and Eihachiro Nakamae. Rendering optimal solar shadows using plural sunlight depth buffers. In *CGI '99: Proceedings of the International Conference on Computer Graphics*, page 166, Washington, DC, USA, 1999. IEEE Computer Society. 21

[WGS99a] Michael Wimmer, Markus Giegl, and Dieter Schmalstieg. Fast walkthroughs with image caches and ray casting. In Michael Gervautz, Dieter Schmalstieg, and Axel Hildebrand, editors, *Virtual Environments '99. Proceedings of the 5th Eurographics Workshop on Virtual Environments*, pages 73–84. Eurographics, Springer-Verlag Wien, June 1999. ISBN 3-211-83347-1. 2

[WGS99b] Michael Wimmer, Markus Giegl, and Dieter Schmalstieg. Fast walkthroughs with image caches and ray casting. *Computers and Graphics*, 23(6):831–838, December 1999. ISSN 0097-8493. 2

[Wil78] Lance Williams. Casting curved shadows on curved surfaces. *Computer Graphics (Proc. ACM SIGGRAPH 78)*, 12(3):270–274, 1978. 13, 17, 20

[WM94] Yulan Wang and Steven Molnar. Second-depth shadow mapping. Technical Report TR94-019, University of North Carolina at Chapel Hill, 1994. 20

[WS06] Michael Wimmer and Daniel Scherzer. Robust shadow mapping with light space perspective shadow maps. In Wolfgang Engel, editor, *Shader X4 - Advanced Rendering Techniques*, ShaderX, chapter 4. Shadows. Charles River Media, Inc., March 2006. 123

[WSP04] Michael Wimmer, Daniel Scherzer, and Werner Purgathofer. Light space perspective shadow maps. In *Proc. Eurographics Symposium on Rendering 2004*, pages 143–151, 2004. 21, 24, 33, 34, 40, 52

Curriculum vitae

Name:	Dipl.-Ing. Markus Giegl
Date of birth:	September 21, 1968 in Vienna, Austria
Nationality:	Austria
Email:	giegl@cg.tuwien.ac.at, giegl@arscreat.com
Languages:	German, English

Education

Sep 1979 - Jun 1987:	Highschool (BRG3 Radetzkystraße); graduated with distinction in Mathematics, Physics, German and English.
Oct 1987 - Jan 1996:	Study of Theoretical Physics at Vienna University of Technology, in parallel to work as database developer.
Jan 1996:	Graduated with distinction in Theoretical Physics on "Automatic Calculation of Green Functions of Arbitrary Quantum Field Theories" done using Wolfram Research's Mathematica at the Institute of Theoretical Physics, Vienna University of Technology.
since Oct 2004:	Doctoral program (Ph.D.) in computer science at the Vienna University of Technology in parallel to job as Community Manager of the European Union GameTools Project.

Jobs

Feb 1993-Oct 1994:	Database Architect & Developer at W.A. Richter; Created AMLV business workflow database solution under MS Access and SQL Server.
Jan 1996–Aug 1996:	Military service in Austrian army (served whole mandatory term of 8 months).
Oct 1996-Nov 2000:	Project Leader and C++ Lead Programmer for role-playing & horror-shooter game Cryptichon[1].
Oct 2000–Jan 2001:	Developed "Moorhuhn 3D" computer game under Ars-Machina in C++.
Jan 2001–Sept 2001:	Project Leader for "FBI Academy" 3D shooter game for Data Becker[2]

[1] http://www.arscreat.com/cryptichon/

[2] http://www.databecker.de/

Mar 2001–Nov 2001:	Authored, produced and promoted a cartoon book parody of Ulrich Strunzs “forever young” German best-seller on jogging.
Oct 2001–Nov 2002:	Lotus Domino Developer at Kendoel Ges.m.b.H.: IBM Lotus Domino development using Java, LotusScript, FormulaLanguage and Javascript.
Mar 2002–Feb 2003:	Developed EOS method to combat Unsolicited Bulk Email (“SPAM”); Drafted & applied for US Provisional Patent (PPA); proof of concept reference implementation in .NET C# and Java.
Jan 2004–Sep 2004:	System Architect at Topcall International: Design and implementation of flexible setup and installation system for workflow solution mostly employed in telecommunications and banking industry.
Oct 2004–May 2007:	Community Manager in The European Union GameTools Project[3].

[3] http://www.gametools.org/