MAGISTERARBEIT

An Adaptable Real-time Soft Shadow Algorithm for Multi-lit Semi-static Scenes

ausgeführt am Institut für
Computergraphik und Algorithmen
der Technischen Universität Wien

unter der Anleitung von
Univ.Prof. Dipl.-Ing. Dr.techn. Werner PURGATHOFER
und
Dipl.-Ing. Dr.techn. Michael WIMMER
als verantwortlich mitwirkenden Universitätsassistenten

durch
Olivier TOROMANOFF
Boterstraat 21, NL-3511 LZ Utrecht

Utrecht, 18. Oktober 2005

Unterschrift

Table of Contents

I. Introduction

Casted shadows are a crucial topic in 3D computer graphics. They have a major influence on the realism and on the quality of rendered pictures. Moreover, without them, the relative depth of the objects in a scene can often not be understood. But rendering high-quality shadows in real time is a challenging problem.

This report deals with the work done during an internship at Navigram BV. This company provides online 3D interactive solutions to the furniture and real estate industries. Their products are based on a rendering engine that does not support shadowing. The aim of this internship was therefore to implement shadowing in a real-time engine and this report explains how we did it.

We will first describe the context of this master thesis and discuss the specificities that influenced our work. Then we will make a short introduction to shadowing and define the vocabulary used. The chapter IV consists in an overview of state-of-the-art shadowing methods. Using some of the presented algorithms, we implemented the first shadowing methods. They will be discussed in the fourth part of this report. But these basic techniques did not provide the aimed level of quality so we had to improve them later on. Finally, we will present the final rendering pipeline that is able to render high quality shadows in real time by combining the various algorithms.

II. The context of my master thesis

In this chapter, we shortly present Navigram BV and its products. Reviewing the requirements of Navigram will give us a clear idea of the most important requirements of the final engine. These considerations lead our development process and influenced the theme of this master thesis.

1. Navigram BV, the company and its products

The work that I describe here was done during a six-months internship I did at Navigram BV, a small company based in Utrecht in the Netherlands. This firm is specialized in interactive 3D design solutions for the housing and interior industry. Currently with four full-time and three part-time employees, it offers design and visualization tools for the non-technical user, essentially meant for a web-based deployment. The main asset of Navigram BV's software is its ease-of-use. Thanks to a very intuitive interface, the handling is very quick to understand, even for an average user. Originally developed for the Dutch market, the products of Navigram BV are now distributed in several other European countries.

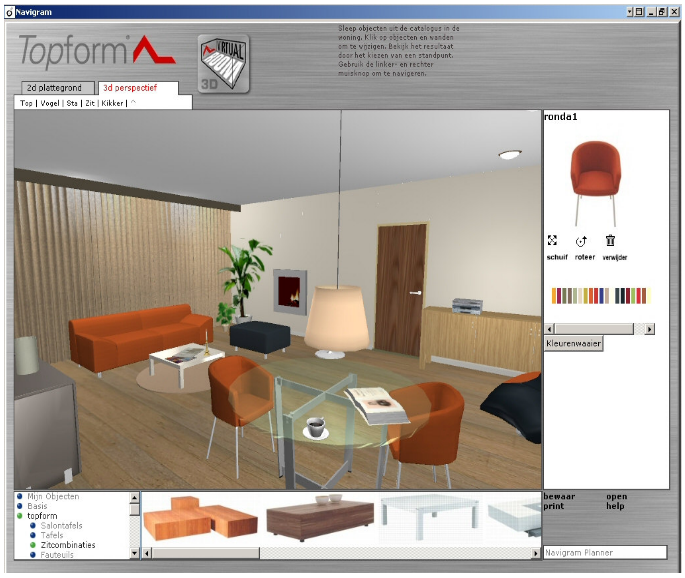

Sleep objecten uit de catalogus in de woning. Klik op objecten en wanden om te wijzigen. Bekijk het resultaat door het kiezen van een standpunt. Gebruik de linker- en rechter muisknop om te navigeren.

Figure 1: Screenshot of one of the Navigram products

Although they are based on the same core product, the different areas of application of Navigram lead to different approaches.

Designed for the furniture industry, the Navigram ProductPlanner is a tool to display the products in the company's catalogue in 3D. Its aim is to let the sales staff, shop assistants, dealers help customers plan their interior with the showcased products. The furniture can be fully configured (size, options, colours, material, etc). The interface is a simple drag-and-drop environment in a web navigator window.

In the real-estate industry, Navigram BV provides an effective tool to visualize plans in 3D. It lets the visitor walk through the plan and experience the area. Navigram BV also offers solutions to create floorplans in a minimum time. The result is a floorplan image that can be used as illustration material, as well as a fully interactive 3D model of the house that can be published on the web. The 3D model can be used for walkthrough visualisation and can be linked to the Navigram RoomPlanner, in which one can place furniture, colour walls, select floor tiling, etc.

The last main part of Navigram's activities is aimed at house and home portals. The Navigram RoomPlanner Portal Edition is a unique product that provides visitors of the house & home portals with the ability to interactively decorate their home online in 3D using products of various suppliers.

Interactive demonstrations of the products can be found on the company's homepage: www.navigram.com under the link “Online demos” (currently, Internet Explorer is the only one supported navigator).

2. The existing graphical engine

All of these products are aimed at providing a 3D representation in real time on the web. Therefore, an important part of Navigram's solutions is the visualisation part. This end-user part is done by an ActiveX plug-in called PageDive Viewer. Integrated in an HTML page via JavaScript commands and interactions, it controls the 3D display and enables interaction through the HTML interface and the navigation in the 3D environments through mouse control. This plug-in was developed in Delphi.

The graphical engine in itself is a small DLL that is downloaded on the user's computer and run locally. It is DirectX-based. This application is controlled by the Delphi application with basic orders such as “draw a certain model at a certain place”, “set a light parameter to a certain value” etc… In fact, the DLL is just a very basic rendering environment that provides the main application with feedback on the user's actions. It does not handle anything else than the 3D display. All the management of the virtual universe is done by the Delphi application. The company does not own the sources of the actual engine, which is why I had to develop my work from scratch.

This engine is very classical and handles only basic features. For example, it supports shading, transparency and lighting but does not include advanced features like shadowing, bump-mapping or global illumination. My task at Navigram during this internship was to develop a proof of concept for extra features that were considered for future releases. My tutor let me work very independently and I had to choose the rendering process that I wanted to improve by myself.

I determined that the major missing feature was shadowing. Shadows are very important to render more realistic pictures. Moreover, in this kind of drag-and-drop interaction, the user has to understand the relative position of the objects, which is often impossible to do without shadows: shadowing is therefore crucial to the ease-of-use of the software. Shadows also greatly contribute to making more aesthetic pictures, a feature which is essential for a sales support program.

3. The specificities of the engine

One possibility is to develop an engine capable of rendering any kind of configuration. But in our case, our implementation will be specific to the Navigram renderer. This environment has its own specificities which we need to take into account if we are to take advantage of them. Moreover, we will see that implementing shadowing is not an easy task even on current graphics hardware. It is therefore important to identify the simplifying factors and difficulties (from a rendering point of view) of the engine to render the scenes as quickly as possible.

a. Simplifying factors

There are three main simplifying factors to the work environment of the Navigram engine. Firstly, the rendered scenes are interior scenes. This means that we only have to handle furnished rooms. These rooms are composed of walls, floors and ceilings of simple geometries. The pieces of furniture used are existing and future models from the Navigram library. These geometries are mostly modelled by a low number of polygons. Moreover, they are decomposed in as many parts as there are textures, with one model for each part. Thus, the rendered scene will stay quite simple. This will allow us to have low basic rendering times.

Another simplifying factor due to the nature of the handled scene is the simple relationships between the objects. As said before, we will always render the same kind of scenes. This restriction allows us to get some interesting information. Indeed, with only minor modifications to the main program, the rendering engine will have some knowledge on the nature of the rendered objects. For example, it is possible to let it know that a geometry is either a simple scene object, a part of a floor, of a wall or of a ceiling, a light or an outdoor object. These distinctions can be important because the behaviour of the shadows depends on it. Thus, the engine knows intrinsically the objects that are pure receivers or pure occluders and is able to render in a separate process the models that are not affected by the shadowing effects e.g. light sources or outdoor geometries (e.g. skybox).

A last positive factor is that due to the target of the product, the scene is what we could call “semi-static”. Although the user can modify the position, shape and characteristics of the objects and light sources, they usually stay unchanged. Our task is therefore to develop a real-time engine capable handling the changes in the scene. Another crucial part of the rendering process is the start of the application, if we are to respect the real-time condition. Indeed, during the first few frames, even if the objects stay motionless, the constraints will be the same as for a fully dynamic environment. But thanks to this “semi static” property, we will be able to use a rendering process close to an off-line renderer. In fact, some computation could be saved to be re-used in the next unmodified frames. These considerations will help us a lot to implement a real-time shadowing algorithm.

b. Difficulties due to the environment

The difficulties due to the environment also have to be taken in account. Firstly, rendering indoor scenes requires several light sources. Indeed, a room is usually lit by a main light on the ceiling and the atmosphere is then created by adding a few secondary lights. Furthermore, and even though the existing engine does not yet support it, a major part of the lighting of a scene is done by the sun. To get interesting results and nice looking pictures, we would like to add this natural light source to the scenes. Thus, our scenes will be “multi-lighted”. Moreover, a correct approximation of the rendering time for multi-lit cases is to consider that the computation of the shadows produced by N light sources will be N times more costly than the basic process for a single light. Hence, this aspect will imply a considerable performance drop and require an important optimisation of the implemented algorithms.

We must not forget that the future engine is meant to be used by the end-users. These users will access it through their internet browsers on their personal or professional computers. The clients of the furniture or real-estate companies are not all hardcore gamers with state-of-the-art graphic hardware and are even - for most of them - average users with an average equipment which can be quite old. Thus, even if a 100% support is not possible, we have to consider that the engine will be run on very different platforms. Furthermore, due to its integration in a web browser, the engine will not be used as a single application and will often have to share system resources with other programs. So the final algorithm has to be able to use the hardware acceleration possibilities when available and detect their absence on older computers. Moreover, it has to be able to keep a high frame rate during the whole rendering process, adapt to the first configuration found but also to dynamic changes.

To test this adaptability, I worked on two computers and all the results given in this report were measured on them. I mainly used a Pentium 700MHz with 512MB of RAM and a NVIDIA Quadro4 580 XGL (which I will call "slower computer"). What I will call the faster computer is a Pentium 2,4GHz with 512MB of RAM equipped with an ATI Radeon 5800 AGP.

c. The aim of my master thesis

Keeping this analysis in mind, I will now define more precisely what I aimed at in my work. For artistic and ease-of-use considerations that will be discussed more in depth in the following part, I decided to implement the soft shadowing feature. Due to the final use of the engine, i.e. as a basis for a new release of the Navigram renderer, it was necessary to handle multi-lighting. Moreover, the rendering process had to be run in real time or (for slower computers) at an interactive frame-rate. This allowed us to take advantage of the semi-staticity of the scene. Finally, the engine had to give the best possible picture by taking advantage of the hardware acceleration possibilities while still being able to run on an average computer. To achieve this, the algorithm had to be self-adaptable. These considerations led me to choose the following long but descriptive title for my thesis:

"An adaptable real-time soft shadow algorithm for multi-lit semi-static scenes".

III. About shadows

Shadows are a hot topic of the computer graphics research and industry and a specific vocabulary was developed. We will define here the most important terms. Then we will explain why shadows are so important. Finally, we will discuss the criterions relevant to the study of the algorithms and their results.

1. Vocabulary

Shadows result of the interaction of three components: a light source, an occluder and a receiver.

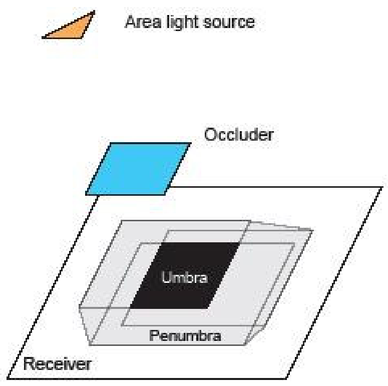


Figure 2: The objects involved in the shadowing process ***(from [19])***

We call "light source" an object that models a primary source of light, i.e. a physical entity that produces luminous energy. We have to distinguish two main kinds of light sources: point light sources and area light sources. All the lights in the real world are area light sources, which means that light is always sent by a surface. But in computer graphics, the model of point light source is widely used and is often good enough to represent reality. In this approximation, the energy of a light source is assumed to be sent from a mathematical point of space. Area light sources are often modelled by a set of point light sources sampling the original light source area. Most of the engines do not support global illumination – i.e. the lighting of the objects by the light reflected on other non emissive objects. Thus the parts of the geometries that are not directly lit by a light source will be fully dark. To avoid this issue, an additional light source is often used: the ambient light. This light influences all the polygons by the same factor regardless of their orientations.

Receivers are all the objects that receive light from the light source. Usually, all objects are receivers. Several situations can occur at the surface of a receiver. A point of the object can be totally hidden from the light source by other objects. In this case, it will not be lit. We call the set of points of the receiver that cannot see any part of the light source the umbra. Other points will be only partially hidden and thus receive light from part of the light source. The points that receive only a part of the sent energy compose the penumbra. The union of the umbra and of the penumbra is the shadow.

The objects that block the whole of or a part of the light energy are called occluders. Occluders cast shadows on the receiver. Because of algorithmic limitations, it is important to distinguish two cases of receiver-occluder relations. The two objects can be different but they can also correspond to several parts of the same object. This means that a surface of an object can be occluded from the light source by another part of the same object. This shadowing process will be called self-shadowing and is often a more difficult case that some algorithms are not able to handle.

2. The role of the shadows in the scene perception

Until the early 90's, shadows were often neglected in real-time applications. However, they are essential to the understanding of a scene. A user will immediately notice if a scene has been rendered without shadows. Shadows are very important for the intuitive perception of an image and are crucial for the comprehension of its composition. A lot of research was done to confirm this affirmation and it was proven that shadows provide the user with useful information to understand more easily the relative positions and sizes of the occluders, the geometry of the occluders and the shape of the receivers.

Sometimes, spatial relationships cannot be determined without casted shadows. For example, in the following figure, the position of the robot relative to the floor cannot be known without the shadow that it produces on this receiver. The Navigram software is based on a drag-and-drop interface whose only feedback is visual. Thus it is very important to provide the user with an image with shadows that will allow him to understand where the currently dragged object is with respect to the ones already placed.

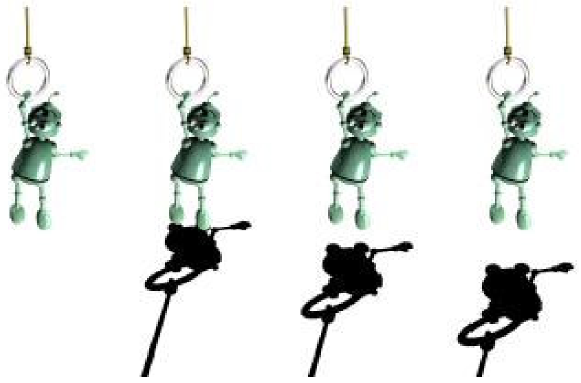


Figure 3: Importance of shadowing to the understanding of spatial relationships. The shadow is here the only clue to determine the robot's position ***(from [19])***

The shape of a shadow is the result of the interaction between two objects. Thus it can provide a visual clue that will help the viewer understand the geometry of a complex receiver or of a partially hidden occluder.

3. Hard vs. soft shadows

As said earlier, the shadow consists in two parts: the umbra and the penumbra. Points of the umbra will only be lit by ambient light and other lights, while points of the penumbra receive a part of the light source energy. But in the case of a point light source, the situation is much simpler: a point of a receiver is either lit or not. In that case, there is no penumbra and we will speak of hard shadows. But in practice, point light sources do not exist and there is always a penumbra surrounding the umbra. Even the sun produces this blur around the shadows. Shadows that are not binary (and therefore more realistic) are called soft shadows.

Rendering fake hard shadows can be problematic: due to their sharpness, they can be mistakenly perceived as objects and thus disturb the user's perception of the scene.

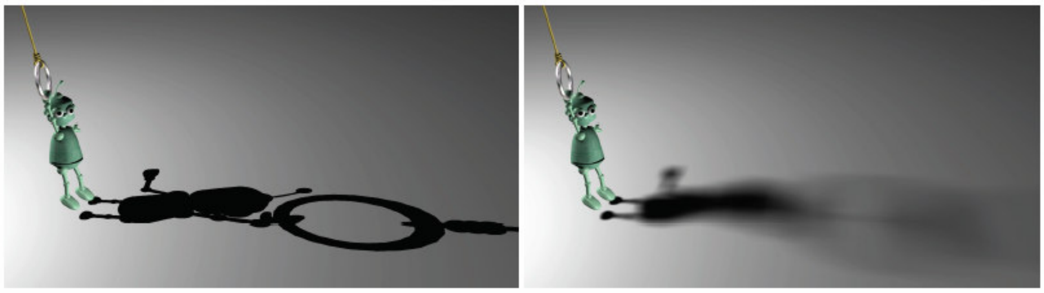


Figure 4: Hard (left) vs. soft (right) shadows ***(from [19])***

The size of the penumbra depends on the size of the light source and on the spatial relationship between the three objects involved. Thus the radius of the softened area of the shadow is related to the ratio between the distance between the light source and the occluder and the distance from the light source to the receiver. Another parameter is the size of the light source. A large light source can cast shadows without umbra. In fact, in the case represented on the following figure, no point of the floor is totally hidden from the light by the robot and therefore each point receives some energy.

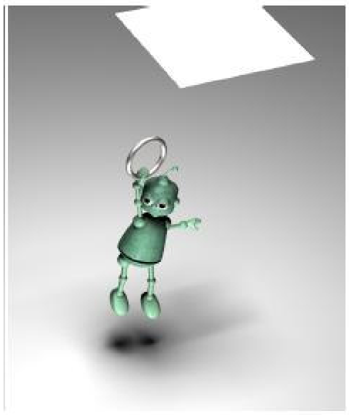


Figure 5: Shadow without umbra casted by a close and big light ***(from [19])***

Humans are used to seeing soft shadows in their visual environment. Thus they will automatically notice if the shadows are hard. A scene rendered only with hard shadows is usually unattractive. As our software is used for marketing support, it has to provide attractive pictures so it is very important for us to be able to handle soft shadows.

4. Some algorithms characteristics

We already mentioned a few features of the various algorithms. Some of these algorithms will be able to handle self-shadowing while others will require a distinction between objects casting shadows and receivers. Another major characteristic is the ability to compute soft shadows. Although it is more costly, it is very important for the realism and the beauty of the rendered pictures. There are two other important criterions.

The Navigram software is meant to be used as an experimenting tool by the user. He has the possibility to move inside the scene and to modify the objects' characteristics and positions. These functions are controlled by the mouse. An interactive frame rate is therefore a must. But to provide the user with an attractive and easy-to-use software we have to render in real time. There is no commonly accepted definition of the words “interactive” and “real-time” but in this study we will require to be able to interact with the scene at frame rates around five frames per second (fps). In a real-time interaction, no reaction latency must be felt by the user which imposes a frame rate of at least 15fps. Of course, the greater the frame rate, the smoother the animation and the more pleasant the software is to use. A major criterion to choose a renderer will therefore be the rendering speed of the shadowing algorithm. These interactivity constraints stop us from using methods meant for off-line rendering such as ray-tracing or global illumination models.

In the following state-of-the-art-study, we will also examine the precision of the rendered shadows. Ideally we aim at rendering physically correct shadows that correspond to the shadows that we would see in the real world. Some of the methods presented here are able to compute real shadows but most of the improvements that allow high frame rates reduce the realism of the result and our aim is to get convincing pictures. Thus we will use the term of “fake shadows” for shadows that are not physically exact but will not trouble the viewer's intuition during his or her perception of the scene.

IV. An overview of real-time shadowing algorithms

This part is a review of state-of-the-art of shadowing algorithms. We will focus on methods that work in real time and exclude solutions such as ray-casting. The first two hard shadows algorithms we will present are the shadow maps and the shadow volumes approaches. Then we will shortly introduce some improvements on these algorithms to render soft shadows in real time.

1. Basic hard shadows techniques

There are two main approaches to computing shadows. The first is image-based and the second object-based. All the real-time soft shadows algorithms are based on these two methods or can be seen as improvements of them.

a. Shadow maps

i. The algorithm

To compute a hard shadow, we have to determine the parts of the scene that do not receive light from the light source. This means that we have to distinguish the points of the geometries that are visible from the light source from the ones that are hidden by other scene geometries. In fact, it is equivalent to a visible surface determination, from the point-of-view of the light source.

The shadow map algorithm is directly based on this consideration. It starts by rendering the scene from the light point of view. During this first pass, we are only interested in the depth value of the rendered points. The result here is the z-buffer which is called the shadow map. This buffer is an array of floats which represent for each pixel the distance between the camera and the closest object at this pixel.

Now we can get the knowledge we need on the visibility of the geometries from the light point of view. We can render the scene and its shadows. We render both the colour and the depth buffer from the view point. The z-buffer will be used during the process for the standard depth culling but also for the determination of the shadows.

Thus, the third step is to perform a test on each pixel of the scene. Thanks to the previous rendering and its resulting z-buffer (the shadow map), we know the geometrical position of the objects with respect to the light. So if the distance between the tested point – the current rendered pixel - and the light is greater than the corresponding distance found in the shadow map, it means that there is an object closer to the light than the current one. We can therefore assume that this point is occluded and that it is in the shadow. So we find which points are lit and which ones are shadowed. Using this knowledge, we can modulate the colour resulting of the lighting model of the current pixel to create the shadows where they should be.

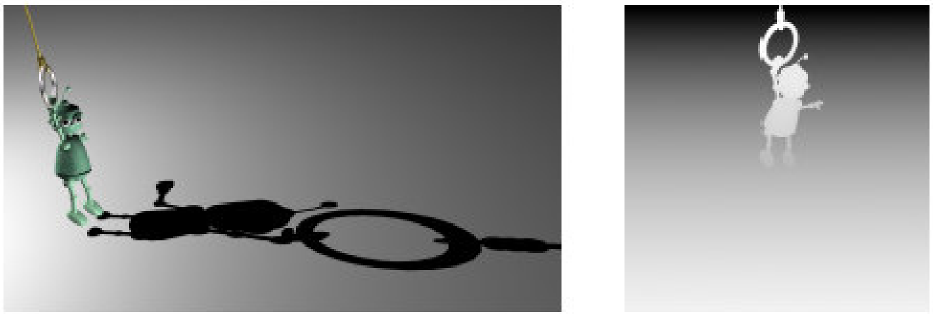


Figure 6: A scene rendered with shadow mapping and the corresponding shadow map ***(from [19])***

ii. Discussion and improvements

Nowadays, shadow mapping is implemented on graphic hardware and there are OpenGL extensions that handle with it. For example, the GL_SGIX_SHADOW can be used to perform the Z comparisons, and the GL_ARB_SHADOW handles the whole shadow mapping process.

The main advantage of shadow maps is that the complexity does not depend on the geometrical complexity of the scene but just on the image size and the number of light sources. Another important advantage is that it can be used with any renderable geometry.

The biggest problem caused by shadow mapping is the aliasing of the shadows. Indeed, shadow maps are computed from the light source and seen from another point of view. If these two points are too far apart, it can cause big distortions and due to the limited resolution of the depth buffer, important staircase effects can occur along the shadow boundaries. A lot of research has been done to deal with this problem and methods to solve it in real time were developed (e.g. deep shadow maps, multi resolution adaptive shadow maps or forward shadow mapping).

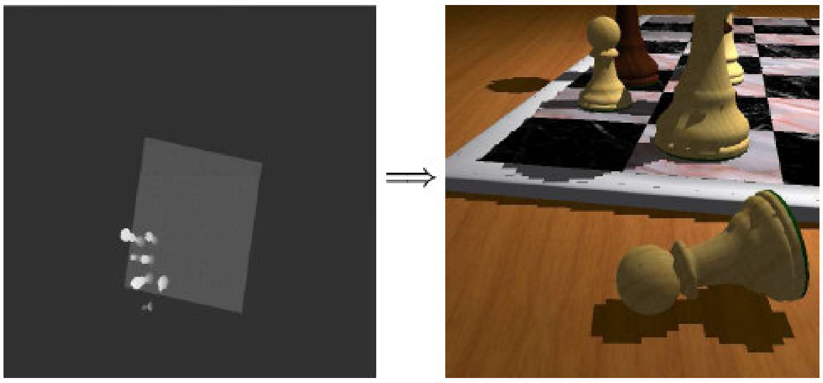


Figure 7: A shadow map viewed from the light location and the resulting aliasing seen from the camera point of view ***(from [20])***

Because it is based on a first rendering from the light point of view, the basic shadow mapping is limited to spot lights. To use it with omni directional light sources, we will have to compute several shadow maps in order to enclose the light source. This modification is known as cube map and increases the complexity by the number of needed additional renderings.

Some artefacts due to the limited precision of the depth-buffer can appear and lead to misplacements of the shadows. Surfaces, for example, can occlude themselves artificially. This sampling problem can be solved by using a small bias to offset the depth buffer.

iii. Projected texture shadows

Another possibility to compute hard shadows with shadow maps is the technique often known as projected texture shadows. This very simple method is quite cheap and therefore often used in video games to get shadows in real time.

To do this, we need to distinguish the geometries that are pure receivers from the ones that are pure occluders. We first render the occluders in black on a white background from the light point of view to get a shadow map. Then, this colour buffer is used as a projected texture to modulate the colour of the receivers. This gives us the shadows casted by the occluders on the receivers. Usually, textures are mapped to the geometry polygons by using the texture coordinates saved in the model. A projected texture does not use this information. In this case, only the vertex positions are taken into account. A relevant metaphor of this process is to consider that the texture is projected in the same way as a picture displayed by a beamer.

The main advantage of this method is that once the shadow map is rendered we do not need special shadow mapping support in hardware. The shadow map is used as a classic texture without any distance consideration on the rendered pixels. As in classical shadow mapping, we can also reuse the result for every frame where the objects do not move. Moreover, this method is not concerned by one of the major issue of classic shadow mapping: the need of a bias to avoid the self-shadowing artefacts due to the limited precision of the depth buffer.

b. Shadow volumes

i. The algorithm

This method offers a purely geometrical way to produce shadows. It was first described by Crow [1]. This algorithm aims to determine the space parts that are occluded. A widely used algorithm is the “Zpass” approach which we will describe here.

We first compute the silhouettes of the objects casted by the light. The silhouette of an object is the list of the edges that separate the lit polygons from the shadowed ones. Then we can build the shadow volumes. These additional geometries are constructed by extruding the silhouette along the light direction. Thus we get polygons defined by an edge of the silhouette, two edges parallel to the light direction and a theoretically infinitely far one. These half-planes define the shadow volume that encloses the area that is in the shadow of the object.

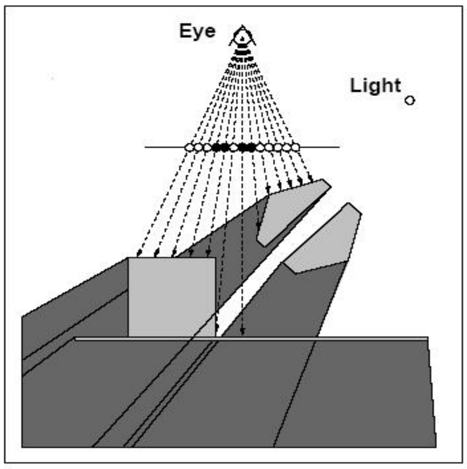


Figure 8: The shadow volumes principle: we test for each pixel if the number of entered volumes is greater than the number of left ones *(from [21])*

The most relevant step of the algorithm is, for each pixel, to determine whether it is inside or outside the different shadow volumes. To know if an object pixel lies inside a shadow volume, we use a simple face-counting algorithm. For each rendered pixel, we count the number of faces that are between the camera and the given pixel. We increase this counter for front-facing faces and decrease it for back-facing ones. If the resulting count is positive, then the pixel is in a shadow volume, otherwise it is lit.

Shadow volumes can be implemented in hardware by using a stencil buffer. The basic face counting method needs two rendering passes of the shadow polyhedron in the stencil buffer. During both renderings, we enable the depth testing but disable the depth and colour writing. For the first pass, we cull the back faces and increment the stencil buffer. Then we render the volumes once more to decrement the stencil buffer with front face culling. Recent OpenGL extensions allow a single pass for rendering shadow volumes.

The final picture is hence obtained in three steps: (1) render the scene only with ambient lighting to initialise the colour and the depth buffers, (2) render the shadow volumes with the previous algorithm to determine the shadowed areas and (3) use the resulting stencil buffer to render the illuminated scene a last time, only where the stencil buffer is not positive. The other pixels that are in the shadows stay unchanged, i.e. only lit by ambient light.

ii. Discussion

Compared to the shadow maps algorithm, the shadow volume approach has the big advantage that it produces shadows with a high resolution. Indeed, shadows are computed with the values of the stencil buffer. This buffer is usually implemented on some bits of the back buffer and has thus the same definition as the produced picture. Another positive aspect of this algorithm is its ability to deal with every kind of light and even with omni directional ones.

The main drawback is a constraint on the rendered geometries. We have to be able to find the silhouettes of the casters. In the classical case of geometries defined by polygons, it is easy to compute, but for other cases (e.g. mathematically defined objects) it can be impossible.

The cost of the shadow volume is linked to the complexity of the original geometries. We first have to perform the geometrical operations of silhouette determination and volume building. The time needed to compute these additional geometries depends on the number of polygons of the models. The same situation arises for the rendering cost, which is linked to the number of faces of the volumes and as said before, there are as many polygons as there are edges in the silhouette. Moreover, during the volume rendering, we render big polygons that have to be filled. Thus the bottleneck of this method is often the fill rate of the GPU.

iii. Problems and improvements

Thanks to its good results and simplicity, this method is widely used, which is why a lot of research was done to improve it. The rendering step can be optimized by minimising the number of rendered shadow volumes. For example, Batagelo [2] uses a BSP tree to do this optimisation. Computing the shadow volumes can be done in different ways. McCool [3] presented an algorithm that computes them from the discontinuities of the shadow maps. The use of programmable graphic hardware (see Brabec [4]) can also be interesting.

But this elegant algorithm has a major disadvantage: it cannot handle the cases where the camera near clipping plane enters in a shadow volume. Indeed, when this configuration occurs, some parts of the shadow volumes are clipped which leads to miscount in the stencil buffer. The final result of the rendering step is incorrect and the generated shadows are misplaced. The method usually used to solve this problem was suggested by Everitt [5] and is called Zfail (in opposition to the previous method called Zpass). The general idea stays the same but we proceed in reverse order. We first increment the stencil buffer by rendering the back faces if they are behind an existing object and then we decrement the count with the front faces that are also behind the scene objects. This solves the near clipping problem but transfers it to the far plan and often requires additional capped volumes.

2. Soft shadow algorithms

The two described methods produce hard shadows. These results are often not realistic enough and insufficient for some applications that use extended light sources. We present here algorithms that are able to deal with soft shadows. They are mainly based on one of the two previous solutions.

a. Image based algorithms, improvement of the shadow maps method

i. Combining several point-based shadow maps

The first method consists in an approximation of an extended light source by several point light sources. In this algorithm, we regularly sample the light area. Then, for each of these sampled points, we compute a binary occlusion map which is the shadow map of a point light source placed in this position. All these resulting occlusion maps will be combined into a single one that will now contain continuous values which enable it to generate soft shadows instead of hard ones.

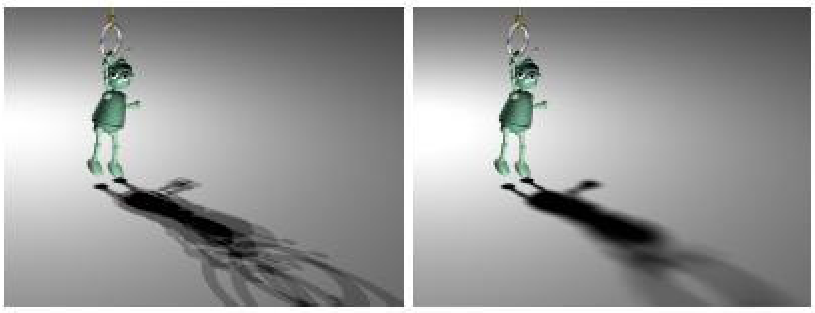


Figure 9: A sampled shadow map with a few samples (4) (left) and the soft result with enough samples (1024) (right) ***(from [19])***

This method is just an improvement on the basic shadow map technique, so it has the same advantages and disadvantages, but here we need N renderings of the scene instead of just one (where N is the number of sample points). Thus the rendering time increases a lot and the choice of a high number of points can stop us from computing shadows in real time. The determination of the number of sampling points is therefore a very important question. It has to be high enough to produce smooth results that do not look like a juxtaposition of hard shadows but it has to stay low enough so the computation can be done in real time. The more sampling points there are, the better the results will be. Using N points gives resulting shadow maps with N grey levels so to get ten grey levels, we will need ten sampling points.

ii. Layered attenuation maps

This method, developed by M. Agrawala et al. [6], is an extension of the previous one. Once again, we proceed with several samples of the light source.

For each sample, we compute the depth buffer seen from the light sample. These results are then warped to the centre of the light source. Using these depth maps, we can find the object that is the closest to the light source for each pixel. This object will be the first layer of the resulting attenuation map. In this map we save the percentage of occlusion for this object. It is given by the number of samples that see this occluder. Then, if other geometries are visible from this pixel, we save them in subsequent layers with their occlusion percentage. Finally, this occlusion map is used at rendering time to modulate the final picture. We are hence able to determine, for each object, its lighting percentage (the previously computed occlusion percentage). When an object is not in the map, it means that it is totally occluded and therefore not lit by the current light.

The memory cost seems to stay reasonable: according to the authors, four layers are nearly always sufficient for moderately complex scenes. But the complexity is once again related to the number of samples used and therefore to the resulting realism.

iii. Quantitative information in the shadow map

This improvement on the shadow map method was first developed by Heidrich [7] for linear light and then modified by Ying [8] to be used with area light sources.

The aim of this technique is to compute a light visibility map. In this map we can find, for each pixel, the percentage of the light area that it “sees” and hence the intensity of the shadow of this pixel. To gain this visibility map, we compute a shadow map for each sample of the light source. For this algorithm, the points chosen as samples are the vertices of the light area. We then find the discontinuities of these maps with an image-analysis algorithm. We build "virtual polygons" that link the objects casting the shadows that produce these discontinuities to the objects that receive these shadows. By rendering these polygons in the viewport of the other samples we compute the desired visibility map. The rendering here is done with a kind of Gouraud shading by giving linearly interpolated values between 0 (for the closest point) and 1 (for the farthest ones) to the rendered points. This visibility map stores the percentage of each edge visible from the light source. Once again, this visibility map is used in a shadow mapping algorithm. We can find the position of each rendered pixel in the visibility map and we can get its lighting with the given value.

The main advantage of this method is the low number of samples needed to get good results. It can be useful to add some samples taken inside the light source to avoid artefacts but we always stay under the required number for a classical sampled shadow map. This algorithm creates perceptually correct shadows although they are fake. The cost of this technique is related to the geometrical complexity of the scene. According to Heidrich and Ying, moderately complex scenes can be rendered at interactive frame rates.

iv. Single sample soft shadows

This algorithm developed by Parker [9] was meant to be used for parallel ray-tracing but was later implemented by Brabec [10] on graphical hardware.

This image-based technique is very similar to the basic shadow mapping. We first render a classical shadow map with its depth value. Then the penumbra is generated by extending the shadow according to the stored z-values. Some implementations create only the outer penumbra and others handle inner and outer shadow gradients.

We generally use the centre of the light source to compute the only shadow map we need. This shadow map is used during the rendering time to get the lighting value of each pixel. We find the position of this final pixel in the shadow map. We then identify the nearest shadow map pixel that is in the opposite lighting state. We assume that we are always in the penumbra of one of the rendered objects. This object can be determined thanks to the previously found pixel. Now that we know the nearest occluder, we can compute the distance to this object and thus, by using an attenuation function that takes the computed distance as a parameter, we are able to set a shadow ratio to the current pixel.

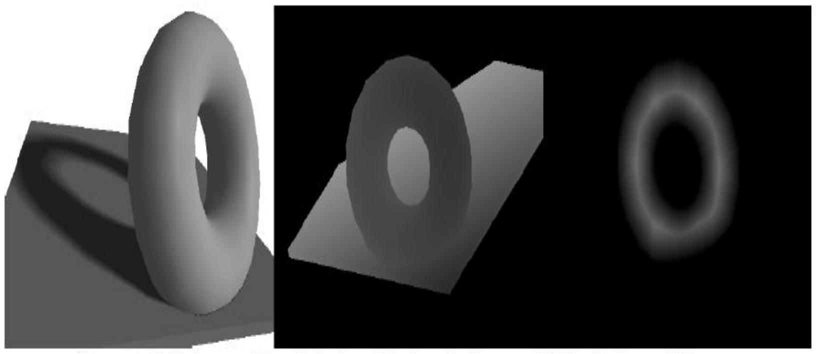


Figure 10: Left: torus with soft shadow. Center: depth-map. Right: shadow-width map *(from [10])*

This algorithm produces fake shadows but they stay very convincing. In fact, we get blurred shadows with a blur radius depending on the ratio of the distances between light, occluder and receiver. This method therefore respects the rules of shadow perception, but because it is based on a single shadow map, it cannot compute physically exact shadows and fails in several cases, for example for a close light source.

The bottleneck of this algorithm is the search for the nearest pixel in the pixel map. The search time can be limited by the choice of a search radius outside which the look for a neighbour pixel will be stopped. This parameter controls the rendering speed and the final quality. The main advantage of this algorithm is its speed. Thanks to the single rendering, Parker and Brabec report frame rates that stay high around 5 to 20 fps on a Pentium 1.7GHz with a NVIDIA GeForce3. A well-chosen search radius and some improvements (such as the look up table presented by Kirsch [11]) can even lead to frame rate up to 20fps (with a Pentium 1.3GHz and a GeForce3).

v. *Convolution technique*

As seen before, the computation of soft shadows is closely related to that of the visibility of the light source from the objects. In the particular case where the light source, the occluder and the receiver lie in parallel planes, this visibility can be seen as the result of a simple convolution. Soler [12] also presented an algorithm based on this mathematical consideration. In this basic configuration, he reported that the soft shadows can be computed by convolving the receiver image with the image of the light source. Although this method is only valid in the parallel and planar case, he developed a method based on an error-driven algorithm that is able to deal with more complex configurations even if the objects are not planar and parallel.

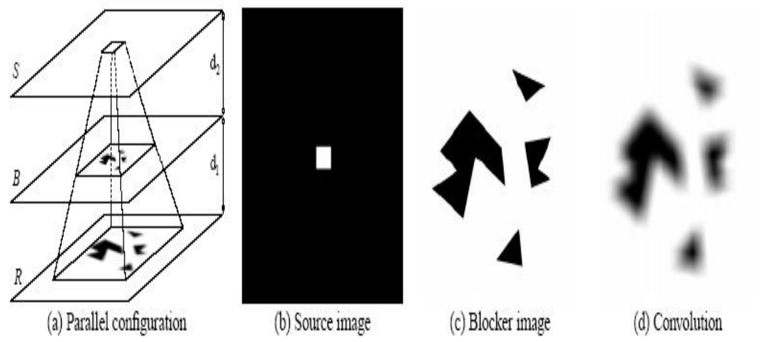


Figure 11: A simple case of parallel light source (S), occluder (B) and receiver (R). The source image is convolved with the blocker image to obtain the shadow map *(from [12])*

Except in the very particular case of parallel and planar light, occluder and receiver, the results are not physically exact but this method always produces nicely looking shadows. The main advantage is the possibility to control very finely the needed computation time. Once the light and occluders images are computed, the complexity of the operation can be fixed to gain the desired frame rate. This algorithm is therefore able to solve configurations with very complex light area shapes. Although it works well for simple cases, the pre-rendering tasks to handle more complex cases can quickly become time-consuming. There are also special cases like elongated polygons in the direction of the light that cannot be handled by the convolution technique.

b. *Object based algorithms, improvements of the shadow volumes*

i. Combining several hard shadows

Once again the simplest and more intuitive way to get soft shadows could be to combine several hard shadows computed by the classical shadow volumes algorithm from several different light samples. We could compute a hard shadow for each of the chosen samples and then average them to gain the desired soft shadowing effect.

But this basic method has the same drawback as the algorithm that combines several shadow maps. By simulating an area light source with N samples, we need N renderings, which means there is a significant performance drop and that it is not possible to use this method for a real-time application. Moreover, the accumulation buffer needed by the averaging process is a feature that is not widely supported on current graphic hardware. Another possible hardware implementation can be done by using floating point render targets that are also not supported by low-end GPUs.

Vignaud [13] developed a variation that could be interesting for cases where the view point stays constant. He proposed that the results of the shadow volumes produced by a light source moving along the original light source area would be combined on the fly. By mixing the resulting pictures in an alpha buffer, it is possible to get an interesting result after a few frames under the condition of viewer motionlessness.

ii. Soft planar shadows using plateaus

This algorithm is the first proposed solution to generate soft shadows by using a geometric approach. It was developed by Haines [14]. It was meant to handle the simple case of a planar receiver.

The first step is a classical shadow volumes algorithm after which we get the hard shadow of the occluders on the receiver. This shadow is saved in a shadow map. To soften this attenuation map that will be used as a modulation texture for the receiver, we build additional geometries based on the edges of the object silhouette. These edges are converted into volumes in the following way. The silhouette vertices are seen as the top of cones whose radius depends on the distance to the receiver. We then create surfaces that are based on the silhouette edges and that join the previously created cones. These volumes represent in fact a kind of shadow volume enclosing the penumbra of the occluder. They are rendered into the shadow map with a colour gradient that simulates the border of the desired soft shadows.

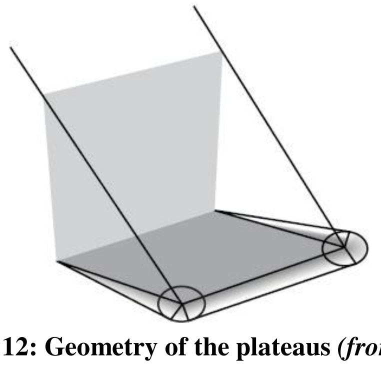


Figure 12: Geometry of the plateaus ***(from [19])***

This algorithm does not modify the hard shadow computed during its first step: it only renders the outer penumbra and produces fake shadows. Another drawback is that it is limited to the simple case of planar receivers. Moreover, it suffers from the same limitation as the original shadow volumes method: the fill-rate bottleneck.

iii. Smoothies

It is a variation of the object-based approach that uses only graphics hardware for shadow generation and uses shadow maps during its final step. It was proposed by Chan [15].

We first render an ordinary shadow map from the light's viewpoint. This shadow map will be used to store the objects depth. Then we construct geometric primitives that we call smoothies at the objects silhouettes. These additional geometries are planar surfaces based on the silhouette edges that are perpendicular to the occluder surface. The smoothies are textured with a colour gradient whose characteristic depends on the distances between the light source, the occluder and the receiver. Once again this method is meant to improve the basic hard shadow map by adding a penumbra around the objects umbra. We render the smoothies into a smoothie buffer and store a depth and alpha value at each pixel. The alpha value depends on the ratio of the distances between the light source, blocker, and receiver. Finally, we render the scene from the observer's viewpoint. We combine the depth and alpha information from the shadow map and smoothie buffer to compute soft shadows.

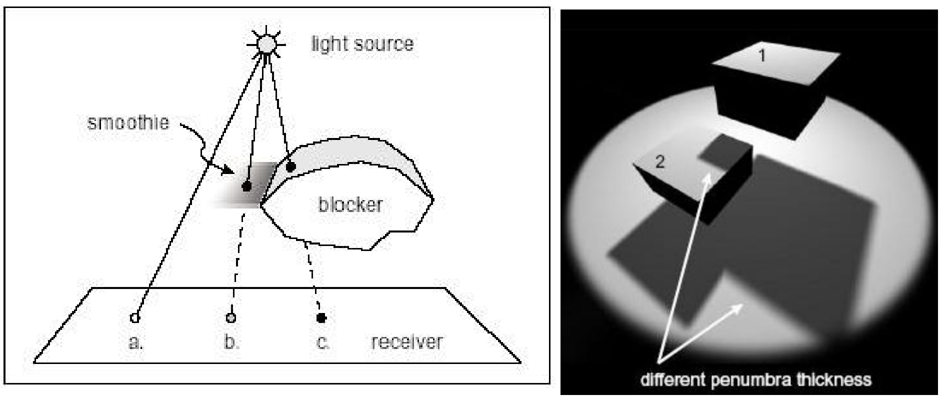


Figure 13: Left: a smoothie built on the occluder geometry. Right: A sample of rendered pictures ***(from [15])***

A limitation of this approach is that computed shadow umbrae do not diminish as the area of the light source increases, so it produces fake (but perceptually correct) shadows. Moreover, this algorithm can produce undesired artefacts because it can be difficult to connect the smoothies correctly at some silhouette vertices. The main advantage is that the frame rate stays high - more than 20 fps on a Pentium 2.6GHz with an NVIDIA GeForce FX 5800 according to Chan - even for complex objects and configurations. This advantage is a consequence of the use of shadow maps instead of shadow volumes. We thus avoid the previous fill-rate bottleneck.

iv. Penumbra wedges

This algorithm is built using the shadow volume method and uses the programmability of modern graphics hardware (fragment programs) to produce real-time soft shadows. It was proposed by Akenine-Möller and Assarsson [16] [17].

The method begins with the computation of the occluder silhouette seen from one light sample. Then, we build for each silhouette vertex a silhouette wedge that is a volume enclosing the penumbra. This volume is based on the silhouette edge and encloses the light area. Then the classical shadow volume is rendered in a visibility buffer to gain the hard shadow. By rendering the front triangles of the silhouette wedges and by using a fragment program we are now able to update the visibility buffer in order to get the soft shadows. The fragment program is used to compute the percentage of the light source visible from each pixel. The last step is to render the final scene by combining the classic rendering process with the visibility buffer.

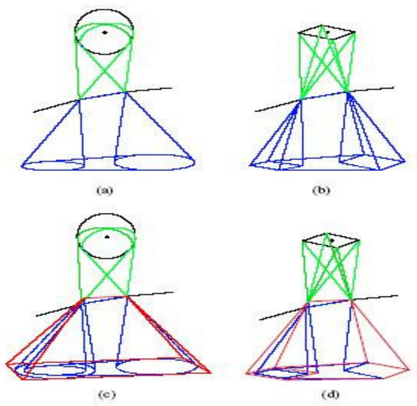


Figure 14: The penumbra wedges *(from [17])*

Two parameters influence the complexity of the algorithm. The first major factor is the number of silhouette edges that is related to the complexity of the objects. Due to the fragment programs that have to be run for each of them, the number of pixels covered by the penumbra wedges is the second determining factor. Thus, an easy way to control the rendering time is to adjust the size of the penumbra wedges. The choice of smaller ones will help to achieve rendering at a better frame rate. Even without this optimization possibility, the main advantage of this method is its speed. Thanks to the use of programmable graphics hardware, Akenine-Möller and Assarsson report very good frame rates (up to 50 frames per second on moderately complex scenes with an AMD Athlon 1.5GHz and a GeForce3). Although the method is based on a single light sample, it is able to produce physically exact shadows for simple cases. To solve correctly more complex situations with large light areas, it is possible to split the original light source in several smaller ones. According to the paper, splitting the lights in four is enough to get a result without any artefacts. Moreover, the number of covered pixels stays the same even if the light sources are split and the splitting does not have a major influence on the final frame rate.

c. Sum up of the presented soft shadow algorithms

Method name	**Type**	**Accuracy**	**Reported frame rate**	**Complexity measure or main bottleneck**	**Hardware requirements**	**Remarks**
Combining shadow maps	Image based	From very bad to physically exact	Need a planar receiver for real-time	Number of light samples	Software implemented	Need to specify the receivers.
Layered shadow maps	Image based	From very bad to physically exact	Interactive frame rates	Number of light samples	Software implemented	X
Quantitative information in the shadow map	Image based	Fake shadows	Few frames per second	Complexity of the shadows	Software implemented	X
Single sample soft shadows	Image based	Fake shadows	Up to 20 fps	Search of the nearest pixel	Software implemented	X
Convolution technique	Image based	Fake shadows	Interactive frame rates.	Complexity of the scene in relation to the basic case	Software implemented	Work well for basic cases (parallelism condition). Need the possibility to break occluders and receivers in subparts. Rendering time easily tuneable
Combining hard shadows from shadow volume	Object based	From very bad to physically exact	No suited for real-time	Number of light sample	Software implemented	Possibility to improve "on the fly"
Plateaus	Object based	Fake shadows always with umbra	?	Fill rate of the graphic card	Software implemented	Need a planar receiver
Smoothies	Object based	Fake shadows always with umbra	Up to 20 fps (50fps)	Complexities of the occluder silhouette	Vertex and pixel shader	Problem with wide light source that have to be splitted
Penumbra wedges	Object based	Physically exact in simple cases	50fps	Complexities of the occluder silhouette. Number of pixel covered by each penumbra wedge	Fragment programs	Weak for complex situations

Table 1 : Features of the presented soft shadow algorithms

V. First implementations

This chapter deals with our very first implementation of shadowing. We developed two basic approaches: shadow maps and shadow volumes. Thus we achieved to get hard shadows at interesting frame rates. Then, we implemented three algorithms to get soft shadows: by blurring hard shadows, by sampling the light sources and with the smoothies algorithm. From a quality point of view, we produced soft shadows from fake to physically exact. We observed a wide range of rendering costs and the best methods cannot be used in real time as they are.

1. The engine

My master thesis at Navigram consists in developing a functional example of a soft shadowing algorithm. The sources of the actual renderer were not available so I had to develop my engine from scratch. According to Navigram, many users experienced problems while running the previous engine that was based on OpenGL. That is why the current one is DirectX-based and why I will concentrate myself on this API. My first task was therefore to develop a DirectX engine capable of handling the basic tasks needed to render a scene, e.g. load geometries and textures and then render them with lighting. All the interactions are handled by the command application so I only developed the basic user interactions that allow the walkthrough with mouse and keyboard. To be as compatible with users' installations as possible, I chose the version 8 of the Microsoft DirectX API and developed the application in C++ with Visual C++ 6.0.

The software that I developed is divided into two parts. The most important part corresponds to the future DLL that takes care of the rendering part. I made this separation in case of a future real deployment. As written before, the graphical engine is controlled by a Delphi application which gives simple commands. For my study, I wanted to have a stand-alone environment, so I developed a second part too. This part is defined as the DLL command application. It consists in several classes that are able to read files containing scene descriptions and generate the instructions that have to be sent to the DLL. It also provides debugging facilities.

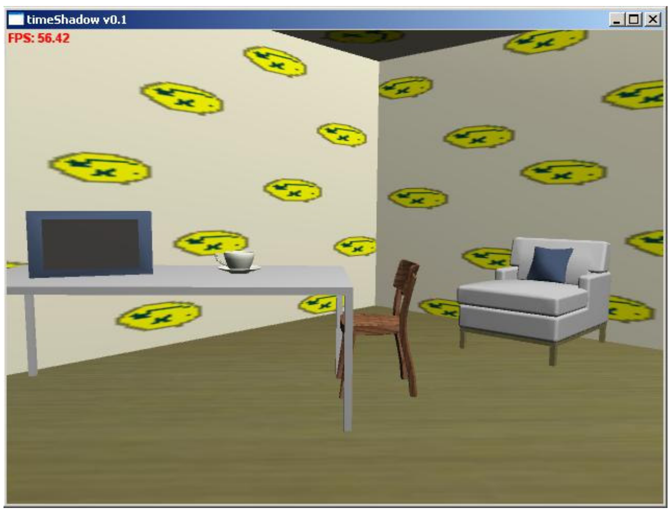


Figure 15: A screenshot of our test scene rendered with the basic engine. The presented point of view is the one that will be used for the performance measures.

2. Projected texture shadows

a. The algorithm

Following the review of the shadowing algorithms done previously, I chose to implement the easiest and fastest algorithm first. This is why I began with the implementation of projected texture shadows. This algorithm is interesting because it only needs one additional rendering pass. It has a main constraint: we have to distinguish the occluders from the receivers. As written before, it is the case in our environment that we know which objects are casting shadows on the floor and on the walls, which are the main receivers. I was not able to implement real shadow maps based on depth comparisons because my GPU did not support any of the hardware accelerated features like pixel shading needed to let the graphics processor compute it.

We start by rendering off-screen all the objects that cast a shadow in black on a white background from the light point of view. This kind of rendering is done in DirectX by using a texture defined as a RENDER_TARGET. This definition allows us to store the content of the back buffer in a classic texture. We first declare this texture, then we attach it to the back buffer. This means that the rendering in the colour buffer will actually be done on the texture pixels as well. After detaching it, this texture will contain the content of the back-buffer. This surface can finally be handled as a texture. Thus, the generated shadow map is used during the final rendering stage as a projected modulation texture for the floor and for the walls.

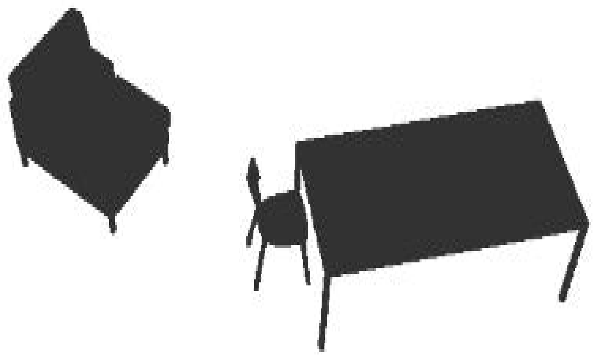


Figure 16: A shadow map

The algorithm runs as follows:

```
Attach the shadow map to the back buffer.
Set the viewport at the light position.
Set the rendering parameters to render in black without texturing and lighting.
Clear the rendering buffers.
Render the occluders.
Detach the shadow map from the back buffer.
Clear the buffers.
Set the viewport at the viewer position.
Render the occluders with texturing and lighting.
Render the receivers with their texture at the first texture stage modulated by the
     shadow map at the second texture stage.
Show the back buffer.
```

Algorithm 1: The projected texture shadows algorithm

b. Results and discussion

This algorithm allows us to get very good frame rates. On the slowest computer we rendered at more than 40 frames per seconds even though the engine was not optimised and the shadow map was recomputed for every frame. This solution is therefore fast. Moreover, the generated shadow maps are only object and light dependent which means that we do not need to recalculate it for every rendering loop when neither the objects nor the light moved.

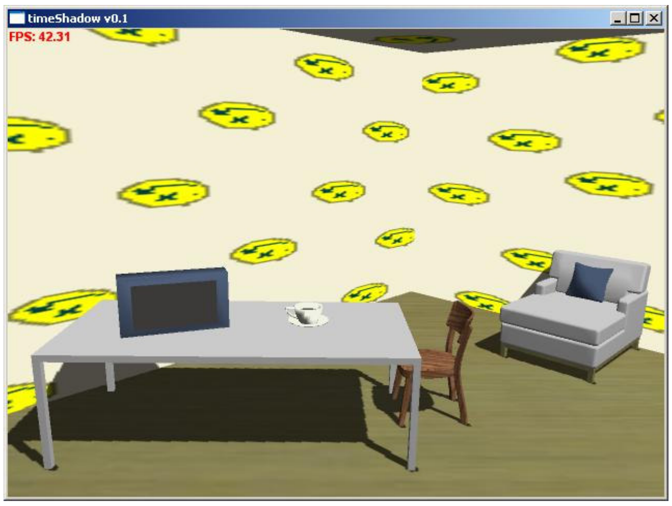


Figure 17: The test scene rendered with hard projected texture shadows

However, this solution produces very basic hard shadows. This algorithm does not support self-shadowing and only computes the shadows casted by the objects on the receiver. In the case of big distortions, the shadows can be blocky too. Furthermore, the chosen basic implementation handles only spot lights.

3. The shadow volumes algorithm

a. The algorithm

I then implemented the second basic technique: the shadow volumes. Shadow volumes are geometrical constructions built on the models. They represent the portions of space that are occluded from the light and allow us to compute in the stencil buffer the parts of the final picture that are shadowed. First, the algorithm finds the silhouette, i.e. the list of edges that separate light and shadow. This silhouette is then extruded along the light direction to build the volumes. The volumes that can be seen in green on the picture are the shadow volumes.

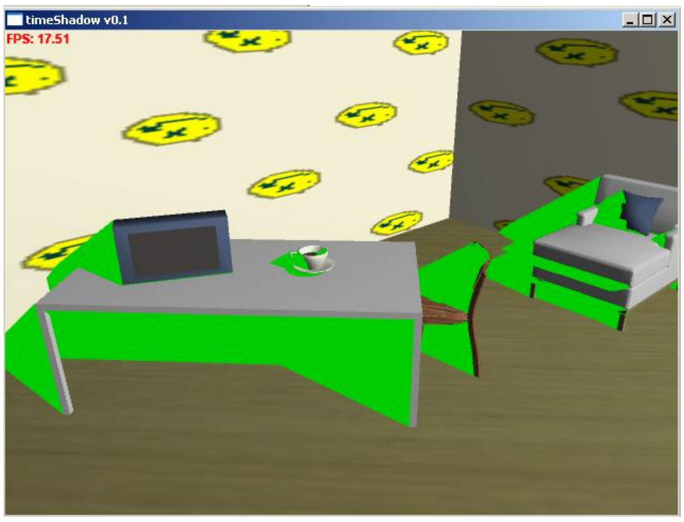


Figure 18: The shadow volumes of the scene are shown in green

After an initialisation of the depth buffer provided by a rendering of the scene objects, we first render the front faces of these volumes by incrementing the stencil buffer and decrease then the count with the back faces. The pixels of the stencil buffer where the count is positive are the shadowed ones. I first wrote a simpler version of this algorithm where the shadows are approximated. In this method, we render a big transparent grey square in the front of the camera where the shadows are as pictured in the following drawing. This process darkens the shadowed areas and allows us to save a scene rendering.

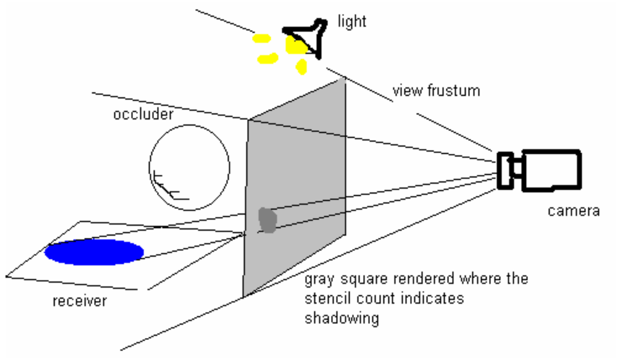


Figure 19 : Shadowing via a transparent grey square.

The pseudo code of the implemented algorithm is therefore:

```
Compute the edges lists that compose the silhouettes of the objects.
Build the polygons that compose the shadow volumes by extruding the silhouette
     edges along the light direction. An edge will be replaced by a polygon whose
     vertices are the two edge vertices and the extruded corresponding points.
Render the scene in the colour- and z-buffer.
Disable colour and z writing.
Render the shadow volumes in the stencil buffer by incrementing it.
Invert the polygons culling.
Render the shadow volumes in the stencil buffer by decrementing it.
Disable zTesting and enable colour writing.
Render a transparent grey square where the stencil buffer stores positive values.
Show the back buffer.
```

Algorithm 2: The shadow volumes algorithm

b. Silhouette computation

The first step of the algorithm is the silhouette computation. We call silhouette of an object with respect to a light the list of its edges that separate the lit polygons from the non lit ones. I first implemented a non-optimised version of this process. In this method (applicable to every geometry), all the edges of the object are tested and added to the final list when needed:

```
Clear the result list.
For each polygon pi:
  If pi is lit:
    For each edge ei of pi:
      For each polygon pj:
        If pj is not lit:
          For each edge ej of pj:
            If ej=ei:
              If ei is not already in the result list:
                Add ei to the result list.
              End if.
            End if.
          Next ej.
        End if.
      Next pj.
    Next ei.
  End if.
Next pi.
```

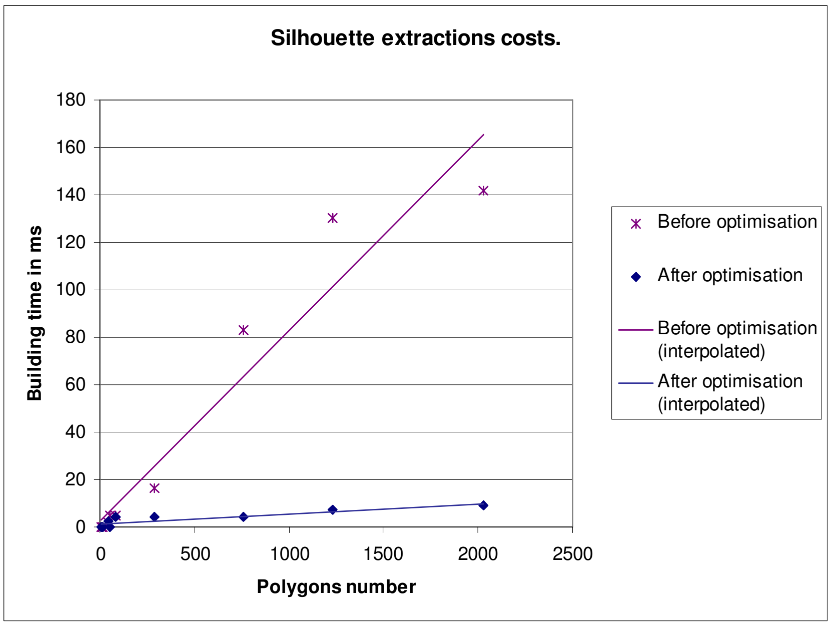
Algorithm 3: The silhouette computation algorithm

As we can easily see, this method needs a lot of nested loops and is therefore very time expensive. Moreover, we need to be able to determine precisely if two edges are equals, which can be problematic because of numerical round-off errors. To improve this process, I proposed a new file format where the connectivity of the models was saved. This format uses connected geometries. We store information to determine the neighbours of each polygon. This optimisation allows us to use another algorithm where only one loop is needed:

```
Clear the result list.
For each polygon pi:
  If pi is lit:
    For each neighbour pj of pi:
      If pj is not lit:
        Store the common edge of pi and pj in the result list.
      End if.
    Next pj.
  End if.
Next pi.
```

Algorithm 4: The silhouette determination algorithm for connected geometries

We can see on the following graph that this second method leads to lower computations costs and reduces the computation time by an important factor for the most complex geometries.



Graph 1 : Comparison between the needed times for the two silhouette determination algorithms

c. Results and discussion

This technique gives very accurate hard shadows and it enables self-occlusion (for example the shadow on the chair). The frame rate stays low (here around 17fps and 41fps on the quicker computer). Once again, some computations such as the volumes only depend on the light and the object position, so it could be possible to avoid recalculating them at each frame. This method has the advantage of being applicable to any kind of light. The use of directional, spot or omni directional lights will only affect the process at the extruding step by modifying the way we extrude the silhouette points.

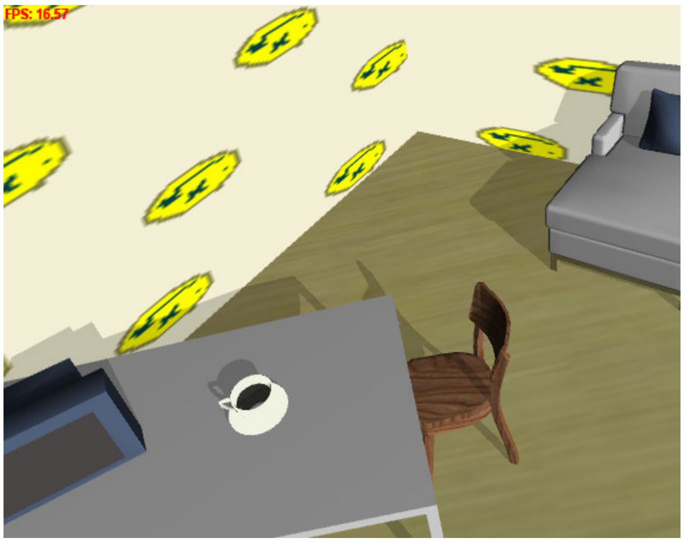


Figure 20: The scene rendered with shadow volumes

4. Blurred projected texture shadows

a. The algorithm

After working on the two basic shadowing algorithms, I began to try to get soft shadows. The first method that I tried was a very basic technique to get fake shadows with not so sharp boundaries. I thought that it could be interesting to improve the very effective projected texture shadows just by blurring the shadow map. The blurring process is done by mixing N^2 times the same shadow map with a different space offset using the following algorithm:

```
Compute a shadow map with the method of the projected texture shadows.
Clear the back buffer.
Disable z writing and testing.
Enable alpha blending.
Attach the final shadow map SMf to the back buffer.
For i=1 to N:
  For j=1 to N:
    Render a square with the shadow map as texture, with an alpha value of 1/N²,
     with the i-th horizontal offset and the j-th vertical one.
  Next j.
Next i.
Detach SMf.
```

Algorithm 5: The blurring process

We have to set two parameters: the number of steps N and the desired radius r of the blur. These two parameters allow us to compute the offset values (which will be r/N). In our implementation, we chose a value of 6 for the parameter N. This value is big enough to produce a smooth blur. Then, like in the original algorithm, we render the receivers with this blurred shadow map as modulating texture.

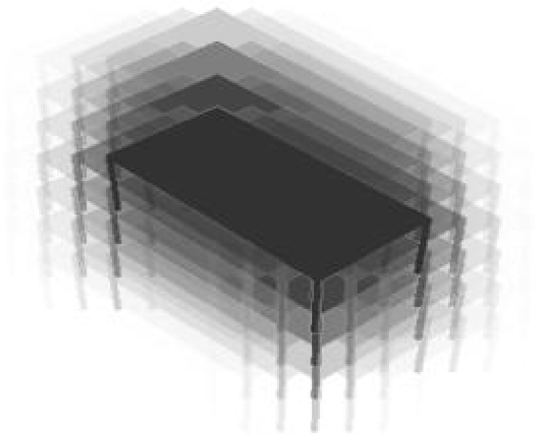


Figure 21: A exaggeratedly blurred shadow map

b. Results and discussion

This method is only a simple improvement of the previously discussed algorithm. It has therefore mainly the same advantages and disadvantages. But it computes soft shadows. Even if its results are not physically correct, the computed pictures seem more realistic. Moreover, with a relatively low additional cost, it stays a cheap solution.

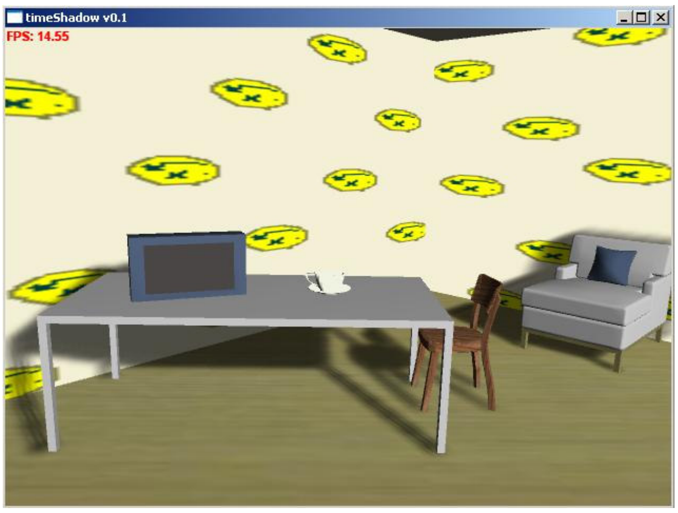


Figure 22: The scene rendered with blurred shadow maps

5. Sampled projected shadows

a. The algorithm

The second way to soften the shadows is the most simple to produce physically exact soft shadows. Once again, it is an improvement of the basic projected texture shadows. We mix together several shadow maps instead of just one. Each shadow map was rendered from a different sample point of the light. It is possible to compute the sample's position by several ways but we chose here to approximate spherical lights by sampling concentric circles.

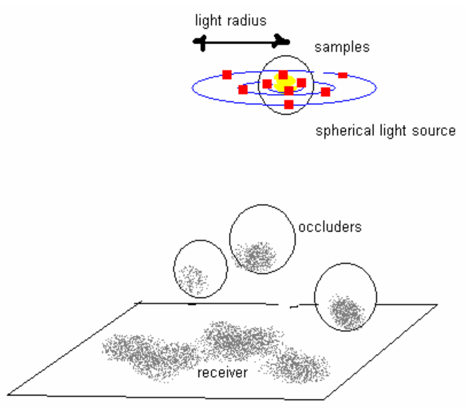


Figure 23 : The used samples location for 9 samples

We had to do an important modification to the previous algorithm. In fact, the way the several hard shadows combine themselves depends on their positions i.e. on the position of the receiver. It was therefore no longer possible to compute a single modulation texture for each light and we had to produce one for each light-receiver couple. We have to introduce a new term: in fact, a shadow map computed and suited only for a particular receiver is called a light map, and that is what we use here. The mixing process is done by setting the camera in front of the current receiver and by rendering the projected shadow maps on it. This way to combine the hard shadows restricts us to the case of planar receivers. Due to the nature of the receivers (walls and floors) it is not a problem in our case. Even the case of not exactly planar walls with small depth variations due to plinths for example, does not lead to noticeable errors and is supported.

The algorithm used to compute the light maps can be described as follows:

```
For each planar receiver ri:
  Clear the shadow map SMi of ri.
  Set the camera in front of ri.
  Sample N times circles centred on the light position and parallel to the
    receiver.
  For each sample si:
    Attach the back buffer to a temporary texture tTemp.
    Clear the back buffer to white.
    Render the occluders in black from si.
    Detach tTemp.
    Clear the frame buffer.
    Mix tTemp with SMi by:
      Enabling alpha blending, disabling z testing.
      Attaching SMi to the back buffer.
      Rendering ri with only SMi for texture.
      Rendering ri with only tTemp projected with an alpha value of 1/N.
      Detaching SMi.
    EndMix
  Next si.
Next ri.
```

Algorithm 6: The sampled projected shadows generation algorithm. The described renderings are done offline.

b. Results and discussion

We empirically chose to set the number of samples to 20. This number allows us to get interesting results at a reasonable cost. In fact, the produced soft shadows are realistic and quite convincing. Moreover, they are physically exact approximations of the real shadows. But the rendering time needed is very high. Indeed, we need to render the scene Ns x Nr times where Ns is the number of samples and Nr the number of shadowed receivers. The observed frame rates are very low: less than one frame per second on the slowest computer and around 6fps on the quickest one. However, like all the other shadow maps processes, the light maps do not have to be recalculated at each frame. Thus this algorithm stays usable for our engine to render in real time if we optimise it to take advantage of the motionlessness of the scenes.

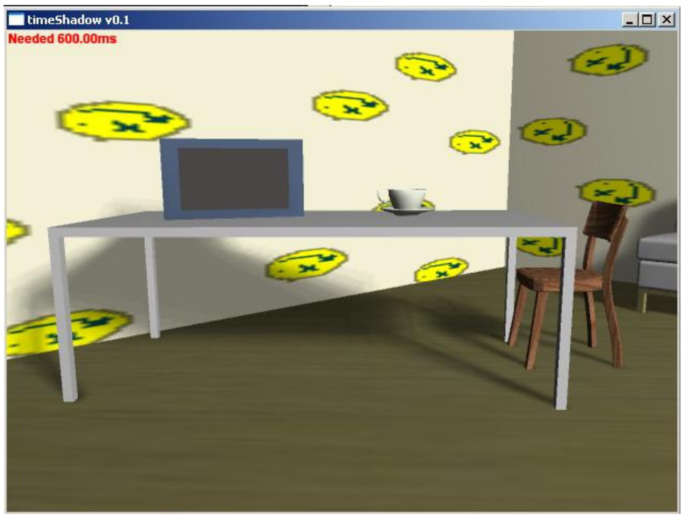


Figure 24: The scene rendered with the sampled projected shadows

6. Smoothies

At this point of the development we are able to render hard or blurred shadows at high frame rates, and physically exact soft shadows at very low speed on the pure receivers. It could be interesting to implement an additional algorithm that would be able to yield better results than hard shadows but at an acceptable frame rate. According to our review, the best candidate to solve this problem seems to be the smoothies method developed by Chan [15]. This technique was meant to produce nice looking fake shadows in real time and is therefore well suited for our problem.

a. The algorithm

Because of the impossibility to use hardware-accelerated depth comparisons, it was not possible to implement the described method, but we combined it with the projected texture shadowing to get soft shadows on our planar receivers.

In this algorithm, we simulate the penumbra of the shadow by geometrical pieces that are called "smoothies". These polygons are built on the silhouette of the object and are filled with a colour gradient. We render them on the shadow map from the light point of view and using their variation of width, we can obtain the blurred border of the soft shadows corresponding to the light distance/receiver distance ratio.

The silhouette of an object is the list of all the edges that are at the limit of the lit part of the geometry. We will extrude this outline with rectangles in order to get the desired effect. Thus we create a rectangle for each edge of the occluder. At all the corners of the silhouette, triangles will be needed to connect continuously the smoothies. The creation process of the smoothie geometries is divided into two steps. First, we build a rectangle with the adapted direction and width for each silhouette edge. Then, we compute additional triangles where they are needed. As the smoothies' rectangles are based on an edge of the object silhouette, we already know one side of this rectangle. The only parameters that we still have to determine in order to complete the polygon are its width and its direction. The final aim of these geometries is to be rendered from the light point of view. Thus we have to find an extrusion direction that is normal to the light direction and to the current edge. The simplest way to get this vector is by performing a cross product between the edge and the light vector. The resulting vector is then normalised. To get the width of the current smoothie, we have to study the geometry of the problem.

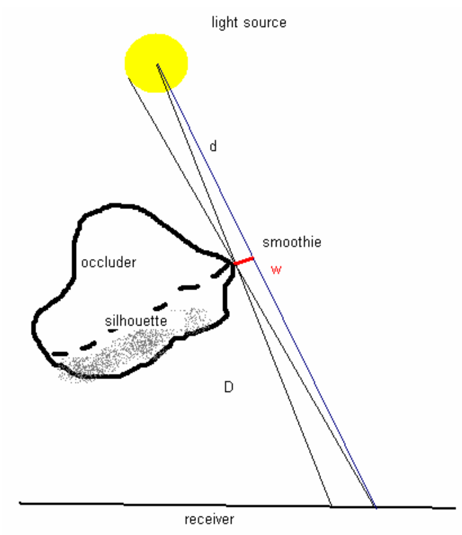


Figure 25 : The geometry of the smoothies

We want to simulate the penumbra of the occluder. The first value that we have to compute is therefore the size of this blurry part of the shadow. In our study, we approximate the light sources by spherical sources. Thus the width of the penumbra is simply related to the ratio between the distances from the occluder to the receiver and from the light to the occluder. As seen in the previous sketch, the penumbra size is the product of this ratio by the light radius. But we have to remember that the smoothies will be rendered from the light point of view with a perspective projection. So we have to compute the width of the smoothies in order to get the calculated penumbra width when they will be projected on the receiver. Once again, we approximate the situation by a classical Thales configuration and after some simplifications, we obtain the final result:

$$width_{smoothie} = radius_{light} \times \frac{D}{d+D}$$

Finally, we add triangles where contiguous rectangles are not connected. These additional triangles are simply built by adding a third edge to enclose the polygon formed by the two following rectangle borders. Lastly, a colour value is set to all the smoothies' points. The ones that are on the silhouette will be rendered with the shadow colour (black in most cases) and the ones of the outside with a fully transparent white. The resulting gradient ensures that we get the penumbra of the casted shadow when we render the smoothies from the light point of view.

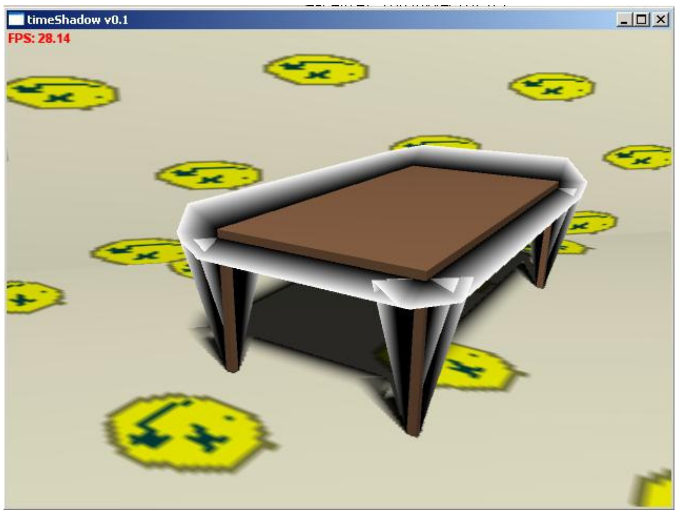


Figure 26: The smoothies of an occluder

The global rendering process of our smoothies algorithm stays very similar to the projected texture shadows technique. We first compute the shadow map by setting the camera at the light place. On this shadow map, we render the gradients given by the smoothies' geometries and the occluder in black to get the umbra. The resulting texture is then used as a modulating texture to render the receiver.

b. An optimisation

As we can see in the formulae, the geometrical characteristics of our implementation of the smoothies method depend on the relative positions of the three objects involved: the light, the occluder and the receiver. This means we have to generate the smoothies of an object for each light-receiver couple, but this generation process is computation-intensive and should be optimized.

Even if we cannot avoid computing several smoothie maps for each light, we can notice that some computation could be saved. We wrote before that three different things have to be determined in order to get the smoothies for a light-receiver couple. We firstly have to find the silhouette of the object with its position relatively to the light source. Then for each of the outline edges we compute the extrusion direction and width. But in fact, the receiver position only has an influence on the last step of the process. Contrary to the width that depends on the relative positions of the light, of the occluder and of the receiver (cf. the previous equation), the silhouette and the extrusion direction only depend on the light and the occluder positions. Thus, a major improvement could be done by splitting the generation algorithm into two parts: a first pass for every light source for the object silhouette and the extrusion direction only, during which the values that are only light-dependent are computed. Then we compute the extrusion width for each receiver which will allow us to build the final smoothies geometry. The shadow maps computing algorithm is then:

```
For each light source li:
  Find the object silhouette
  For each edge ei of the silhouette:
    Compute the extrusion direction di
  Next ei
  For each receiver ri:
    For each edge ei of the silhouette:
      Compute the extrusion width wi
      Build a rectangle with the edges ei, di and wi
      Set the correct colour values to the smoothies points
    Next ei
    Set the camera at the light source position
    Render the object in black on the receiver shadow map
    Render the smoothies on this shadow map
  Next ri
Next li
```

Algorithm 7: The optimised smoothies computation algorithm

c. *Results and discussion*

The smoothies algorithm produces nice looking fake shadows. Even if they are often too dark because they always have an umbra, they approximate real shadows well enough to be used as they are. In all the cases, the results are incomparable with the ones given by the basic projected texture shadows method. We obtain here shadows that simulate the penumbra and its thickness changes.

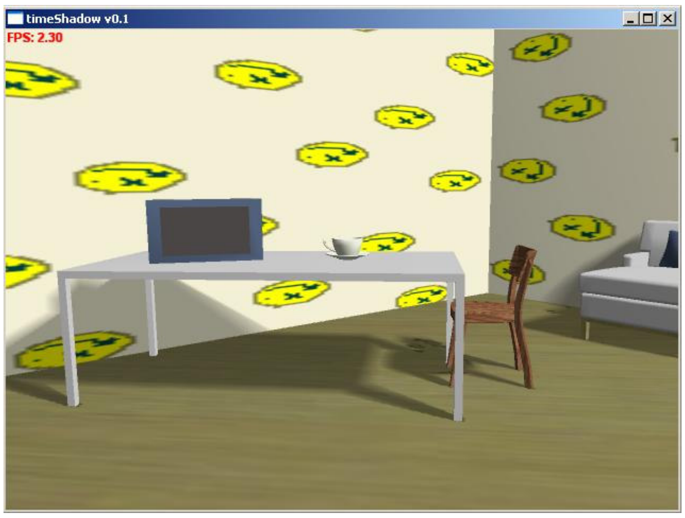


Figure 27: The scene rendered with smoothies

For the first implementation without the presented optimisation, we measured frame rates around 5fps. These frame rates are not enough for a real-time use but once again we have to consider that a calculated shadow map can be re-used when the objects do not move. Although this method is based on the projected texture shadows technique and involves the same rendering process, it is much slower. The smoothies' geometries are not so complicated, which means that the process is limited by the computation costs. That is why we developed the previous improvement. With this improvement, we obtain frame rates up to three times higher, which makes this algorithm very attractive.

Furthermore, it is very well suited to our task. Thus, if we recall the whole process, we can see that the most expensive part is the computation step. We first have to get the smoothies of all the objects before being able to render a receiver shadow map. But when all the smoothies are built and an object moves, we only have to re-compute the smoothies of the object that moved. The last step (the rendering part) is still a simple rendering. We now have an algorithm that produces fake but realistic shadows and that is able to update the shadow maps very quickly when needed.

VI. Rendering improvements

The first implementations were not satisfactory. We had to improve them to obtain the final rendering process. In this section, we present the modifications we did to the algorithms. We first worked on the shadow map to get a better resolution, to gain rendering speed and above all to support coloured light sources and multi-lighting. Then we fixed the main problem of the shadow volumes algorithm: by implementing the ZP+ algorithm, we managed to avoid the misplacement of the shadows when the camera steps into a volume. We also improved the rendering process of this algorithm to allow multi-lighting. The result is that our final algorithms produce pictures at a good frame rate with the expected quality. Therefore we combined them and implemented the final rendering pipeline that combines shadow mapping for the pure receiver and handles the occluders with the shadow volumes approach.

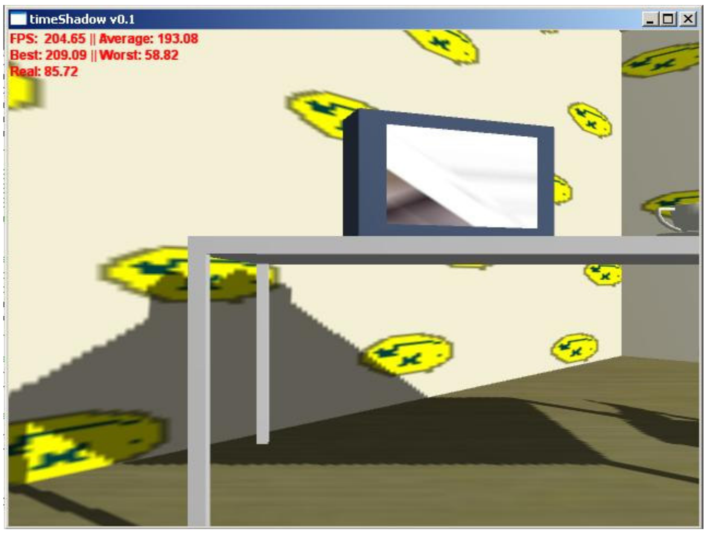
1. Improvements of the shadow maps algorithms

a. Shadow maps resolution

Until now, the projected shadow maps were generated and used with a basic method. We compute the shadow maps from the light point of view by rendering the objects in black on a white background. These shadow maps are then used as projected texture to modulate the receivers' original textures. This projection is performed according to the light view and projection matrix. Even for the smoothies algorithm and for the sampled shadows, we did not optimise the resolution of the results.

But this method has two major drawbacks. Firstly, we need a view frustum centred on the light source. This view frustum and its associated projection is space limited like all view frustums. Thus we cannot simulate an omni-directional light source casting shadows all around itself. This algorithm is well suited for a spot light whose lighting area is restricted. In fact, only in this particular case, we are able to determine a view frustum that includes all the occluders. In the general configuration, it will not be possible to render all the surrounding objects on a single shadow map. The classical way to solve this problem is to use cubical shadow maps. In this widely used technique, we build a cube around the spherical source. Each face of this cube will be a shadow map. Thus the light will be casting shadows in all directions.

The second major problem is the shadow resolution. Indeed, the shadow maps produced by the previous method have a "light resolution". Due to the generation process performed by rendering the objects from the light point of view, we can fix the resolution of the shadow maps seen from the light source. But this is not enough to ensure that we get well-defined shadows. The shadow resolution on a receiver will depend on the position of the receiver. Therefore, if we project a texture along a direction that is almost parallel to the targeted surface, we will magnify its pixels. In extreme configurations, the resulting texturing will be very blocky as pictured in the following figure. A lot of research was done to solve this problem and to get an "image resolution", i.e. to have shadows whose resolution match the camera one.



```
FPS: 204.65 || Average: 193.08
Best: 209.09 || Worst: 58.82
Real: 85.72
```

Figure 28: Aliasing issue for shadow maps

Most of the methods developed to solve these two problems were designed to work in a very general situation. But in our study, we only deal with interior scenes in which the main receivers are walls and ceilings and therefore planar. The implemented solution takes advantage of this and computes a shadow map for each receiver. Moreover, this method allows a future compatibility with the smoothies algorithm. By generating an occlusion map for each light-receiver couple, we solve the problem of non-spot light sources. Thus we are able to render all the occluders of a receiver by chosing a view frustum that contains them. Furthermore, this solution improves the resolution problem. In fact, the quality of the shadows will not be fixed from the light point of view but at the surface level. Instead of a "light resolution", we gain an "object resolution". Aliasing problems can still occur when the camera lies close to a receiver, but they are less noticeable.

We implement this method using the DirectX statement D3DXMatrixPerspectiveOffCenterLH for the projection matrix. This instruction takes several arguments that allow us to define a view frustum with a far plane based on a given rectangle. The origin of the projection is set to the light source position and the far plan to the receiver area. Thus we get a view frustum that includes all the objects that are between the light source and the receiver. These are all the effective occluders that will actually be rendered on the shadow map. This off-center projection is well suited because it is distorted. It allows us to use a view matrix with a view direction that stays perpendicular to the shadowed area and to match far plane and receiver. By setting the far plane exactly on the receiver, we ensure in fact that the back buffer covers exactly the occluded rectangle. We hence get the previously discussed "object resolution" and better results as shown below, even if we do not achieve image based resolution.

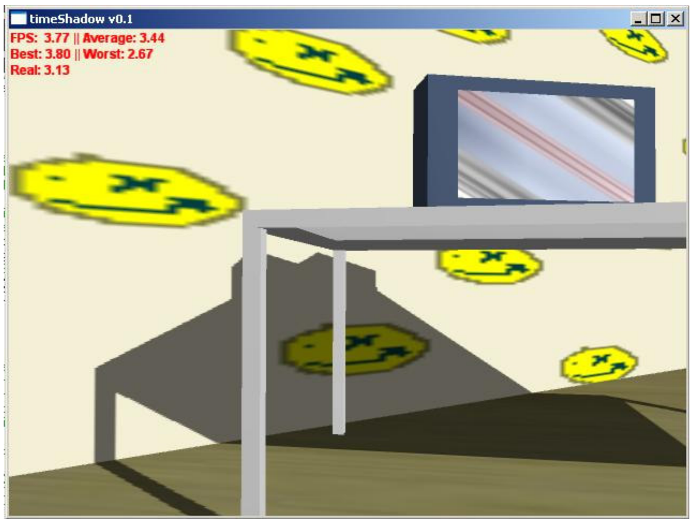


Figure 29: A shadow map with object based resolution. Aliasing is no longer disturbing

b. Shadow culling for the light maps

The projected texture shadow technique and its implemented improvements are interesting in our case because as long as neither the occluders nor the receiver change, the computed modulating texture can be re-used without being recalculated at each frame. Moreover, the modulation process is quite cheap with the current graphic hardware. Thus the most expensive part of this method is the shadow map computation and we would like to improve its speed and to avoid performing it when it is not needed.

In the sampled shadow maps algorithm, we have to render each occluder N times on the modulation texture. For the smoothies solution, the smoothies of each occluder have to be computed and then rendered. Even for the basic projected shadow maps technique, the complexity of the shadow determination stays in line with the number of involved occluders and their complexities. Until now, we considered that every occluder of the scene is a potential occluder for every receiver and so, we take them all into account to get the final shadow map. An intuitive optimisation could therefore consist in generating a list of the objects that are actually casting a shadow on the considered receiver and to limit the generation process to these geometries. Such a list would be interesting because it would save some shadow map re-computations. In fact, we only need to rebuild the shadowing texture of a receiver when the objects included in the occluder list change, disappear or are added. Thus moving a chair located in one corner of a room will not affect the shadows casted on the opposite wall and we will hence be able to use the previous shadow map of this wall without any further consideration for the next rendering loop. The improved shadow map generation algorithm runs as follows:

```
For each receiver ri:
  If the shadow map of ri was never computed:
    Set shadow_Map_Has_To_Be_Computed to true
    Clear the occluders list li of ri
    For each scene object oi:
      If oi casts a shadow on ri:
        Add oi to li
      EndIf
    Next oi
  Else
    For each oi of li:
      If oi has change:
        Remove oi from li
        Set shadow_Map_Has_To_Be_Computed to true
      EndIf
    Next oi
    For each scene object oi:
      If oi has change:
        If oi casts a shadow on ri:
          Add oi to li
          Set shadow_Map_Has_To_Be_Computed to true
        EndIf
      EndIf
    Next oi
  EndIf
  If shadow_Map_Has_To_Be_Computed:
    Compute the shadow map resulting of li
  EndIf
Next ri
```

Algorithm 8: The shadow culling algorithm

This process is based on a function that is able to determine if an object casts a shadow on a receiver according to the given light. As we saw at the beginning of this study, cast shadows are very complicated objects and their determination is not simple. Thus it seems to be unrealistic to try to determine exactly the objects that occlude a receiver. We will therefore implement a more rough approach by approximating the relationships. Our main task is to limit as much as possible the number of items included in the occluder list. Considering too many objects as occluders will not be problematic as long as the number of culled geometries is high enough to compensate for the additional cost. But in any case, our algorithm must not be too strict and eliminate the occluders we do need, which is why the algorithm has to meet the following requirements: low computation cost, ensure that no occluding objects will be culled and eliminate as many occluders as possible.

We met these requirements by implementing a shadow culling process based on the geometry of the bounding boxes: instead of testing the shadow casted by an object, we will test the shadow of its bounding box. Bounding boxes have simple geometries thus we decrease the computation cost. Moreover, bounding boxes are defined as volumes that enclose the original geometry. By definition, they are bigger than the object they contain and no part of the model is outside the cubes so the shadow of an object is always included in the shadow of its box. Thus we will be sure that the rejected objects - i.e. occluders whose bounding boxes do not cast shadows on the receiver - are not occluders for the considered receiver. Testing if a bounding box casts a shadow on a receiver is quite simple in our situation (planar rectangular receivers). We first compute the bounding box of the tested object. Then we project its eight vertices along the light direction onto the receiver plane, which gives us a simple planar polygon and we only have to test if it intersects the receiver rectangle. These projected polygons are pictured in red below:

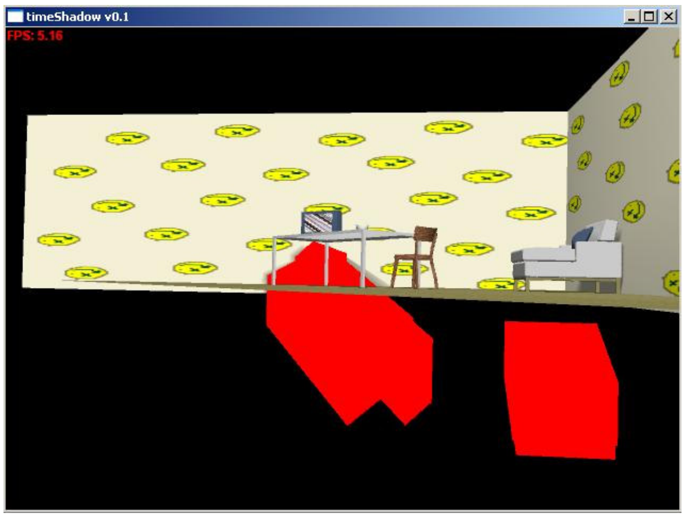


Figure 30 : The projected bounding boxes

c. Coloured light maps

i. Principle

All the previously presented implementations of the light maps are generated and used in similar ways: We simulate the shadows casted by the occluders by rendering them in black on a white background and the penumbra by using grey levels. Then, the resulting texture is used as a modulation texture. This means that the final colour of a receiver pixel will be the texture colour at this position multiplied by the lighting value at this pixel and by the corresponding shadow map value. So a receiver pixel that belongs to a cast shadow will be black. To get more realistic results, it could be possible to use a gradient from a dark colour (matching the colour of the light) to white instead of a gradient from black to white. But in all these cases, this process stays a very rough approximation. In fact, we can easily understand that an occluder will not similarly affect a receiver that is fully lit and a more shaded one. Thus the use of a modulation texture whose colours do not depend on the actual lighting of the receiver can not be effective. So to produce real shadows, we cannot distinguish artificially the lighting due to a light and the shadows that it produces and we will have to combine these two influences in the used texture. Moreover the method used does not handle multi-lighting and coloured lights. For these reasons, we developed another way to get the shadow maps. This technique is closer to the physical description of light and is called light maps.

Instead of computing something that approximates the shadows, we will now try to get the real lighting value at each pixel of the receiver. The first step is the initialisation of the light map. Instead of clearing this texture to white, we will render the receiver geometry only lit by the current light. This will give us a texture whose pixels represent the intensity and the colour of the light actually sent by the light source on the receiver. Then we render the occluders in black and grey on this light map from the light point of view according to one of the previous shadow map generation algorithms. This time, this process is physically correct because an occluder will effectively stop the light received by the receiver. The following figure shows the light map of the floor of our sample lit by a red light.

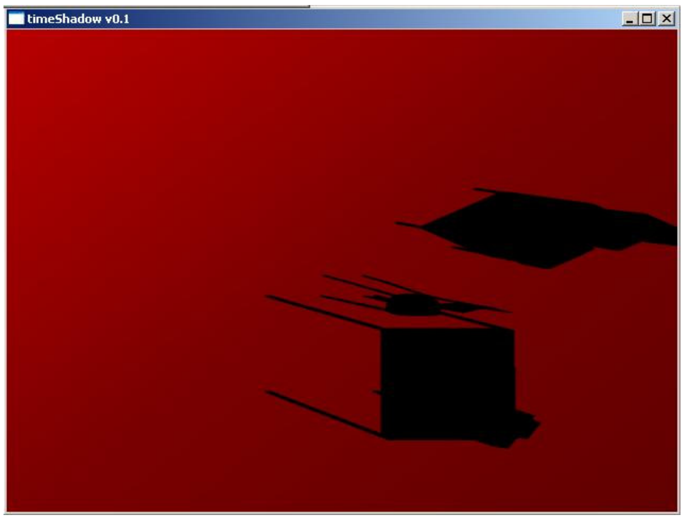
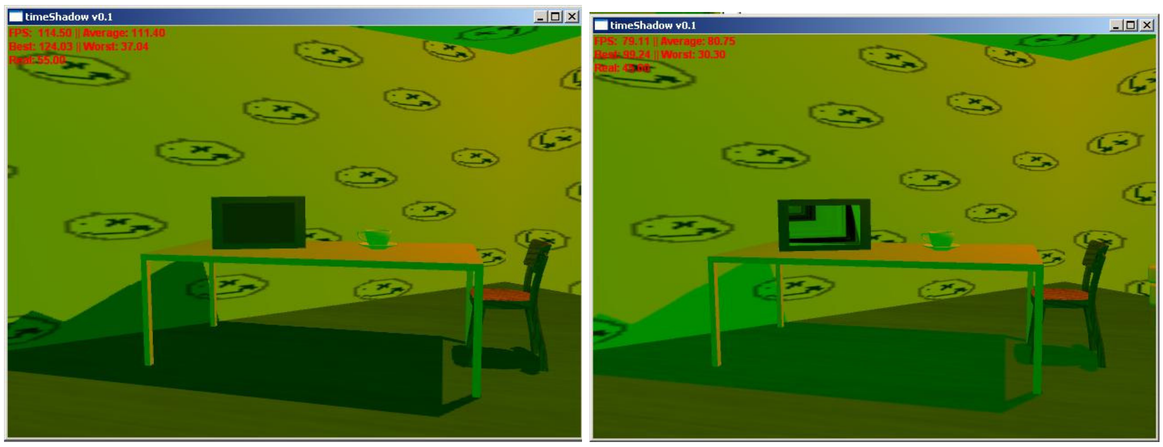


Figure 31 : A coloured light map for a red light source

We have to modify the rendering of the receivers. The resulting texture cannot be used as a modulating texture but has to be added. To render the receiver, we set the texture stages as follows: the result of the first stage will be the sum of the lighting due to the ambient light and to all the lights that are not included in the light map with the light map. Then after this first stage, we have the real lighting values of the receiver area. Then the second texture stage will multiply the result of the previous stage with the texture of the model of the receiver. Thus we actually implement the exact lighting equation:

$$Colour=(I_{currentLight}+I_{otherLights})\times texture=(lightMap+I_{otherLights})\times texture$$



```
FPS: 114.50 || Average: 111.40
Best: 124.03 || Worst: 37.04
Real: 55.00
```

```
FPS: 79.11 || Average: 80.75
Best: 99.24 || Worst: 30.30
Real: 45.00
```

Figure 32 : In both cases, the ambient light is green and the light source red. Left: we use a shadow map: the shadow colour is false. Right: we use a light map: the shadow colour is exact

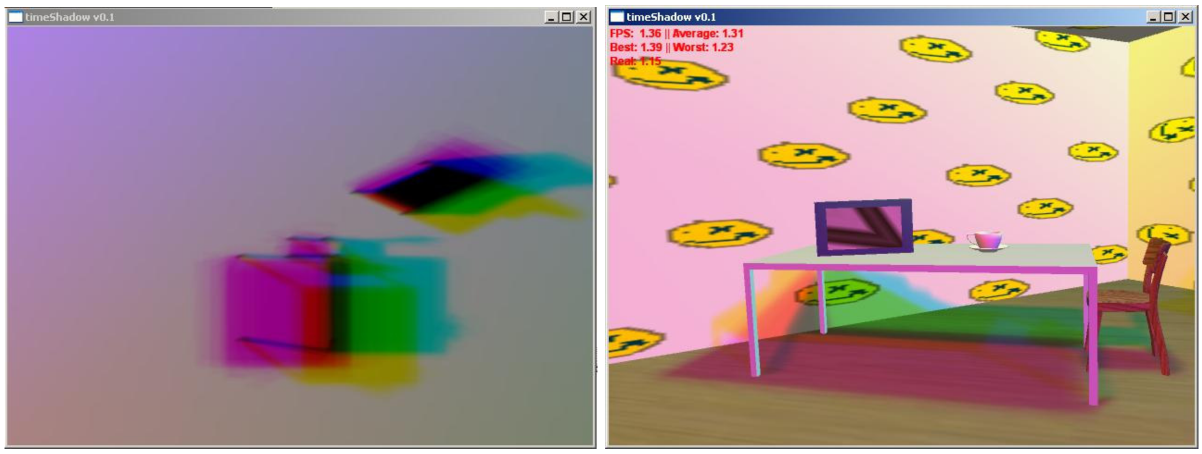
ii. Multi-lighting

To be able to render nice room atmospheres, our engine has to support multi-lighting. This means that the previously described algorithm has to be modified to be able to render a receiver with the occluder shadows generated by several lights. Most of the current low-end GPUs do not support more than two different textures during the rendering of an object. Thus it is not possible to generate a light map for each light and to mix them together using the texture blending stages. But the previous lighting equation is still adapted to our task. We only need to compute a general light map that will sum up all the contributions due to the different lights. We will compute this global light map by using alpha blending. First, one light map is generated for each light source according to the method above. Then we set the camera in front of the receiver and render them on it with an adapted alpha blending. The result will actually be the desired sum.

```
Clear the final light map LMi of the receiver ri.
Switch off the ambient light and all light sources.
i=0.
For each li:
  i++.
  Compute the light map lmi of li.
  Enable alpha blending.
  Set the camera in front of the receiver.
  Attach LMi to the back buffer
  Render ri with lmi as texture and alpha set to 1/i.
  Detach LMi
Next li.
```

Algorithm 9 : The light maps mixing algorithm. The described renderings are done offline.

Finally, the resulting light map will be used exactly as in the single light source case. A sample of a multi-lit light map and of the resulting shadows is pictured bellow:



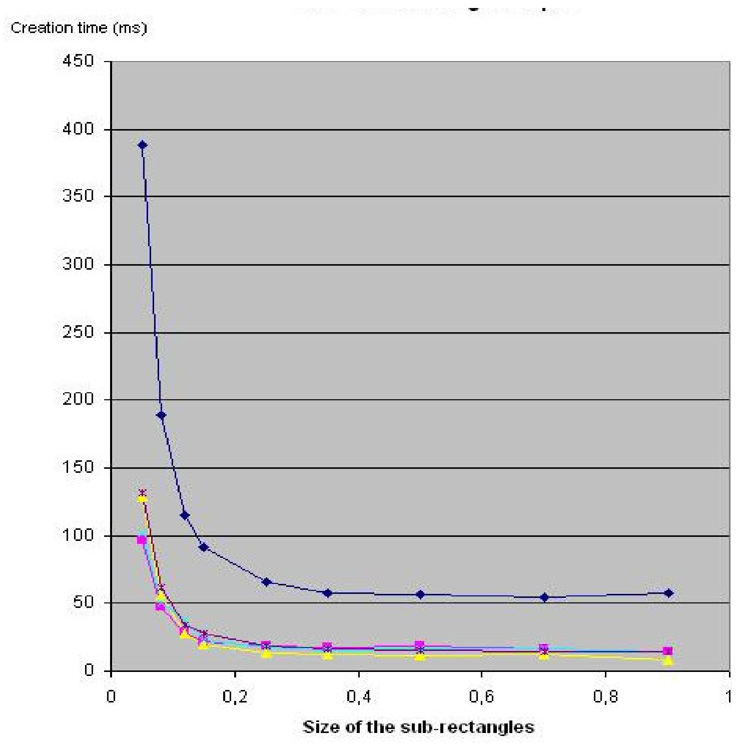
FPS: 1.36 || Average: 1.31
Best: 1.39 || Worst: 1.23
Real: 1.15

Figure 33 : A multi-lit scene with coloured sources rendered with sampled light maps. The light map used for the floor is shown on the left

iii. Fine lighting for the receivers

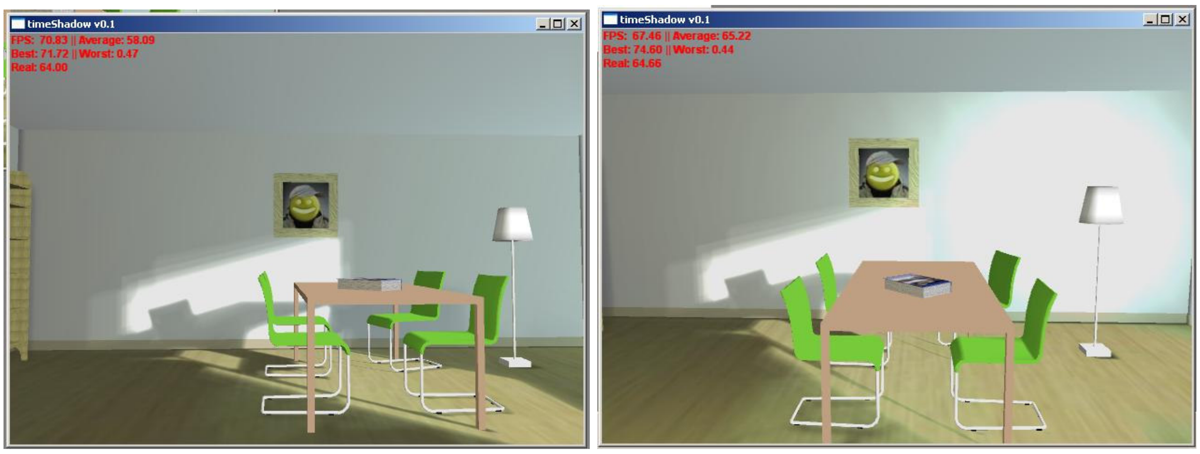
The generated light maps are now realistic simulations of the real lighting process but there is still a problem that causes erroneous lighting values and decreases the beauty and realism of the result. In the previous algorithm we used a rectangle to simulate our receiver to get the light intensity. But DirectX only supports the Gouraud shading lighting model. In this model, the light intensity of each pixel of a geometry is linearly interpolated from that of the geometry vertices, which is calculated exactly. In our case, this method simulates the lighting of a rectangle by a gradient between the four light intensities of the corners. This approximation stays acceptable for small areas but becomes unrealistic for bigger polygons. In our case, the most important receivers are the walls and the floors, which are extensive geometries.

To obtain better results and be able to render the light variations along these receivers more finely, we modified the first step of the light map generation process. Instead of rendering a big rectangle to get the light intensity values, we broke up this big area into a set of smaller rectangles. Thus, instead of a gradient between four values, we get linearly interpolated values from a finer sampling. The following graph allows us to determine the more suited size of the sub-rectangles:



Graph 2 : Computation time of the light map as a function of the size of the sub-rectangles

As excepted we see that the computation time grows very quickly when we reduce the size of the small rectangles. After studying this graph and trying out a few values, we determined that the best value to get visually interesting improvements without a huge frame rate drop was 0,5m. The graphical improvement is shown below where we can see on the right picture produced with the fine lighting process that the variations due to the proximity of the light are now rendered on the wall.



```
FPS: 70.83 || Average: 58.09
Best: 71.72 || Worst: 0.47
Real: 64.00
```

```
FPS: 67.46 || Average: 65.22
Best: 74.60 || Worst: 0.44
Real: 64.66
```

Figure 34 : Left: without fine lighting. Right: with fine lighting

2. Improvements of the shadow volumes algorithm

a. The ZP+ algorithm

i. The algorithm

The basic shadow volumes algorithm is a very powerful method to get the shadows of a scene. It allows us to solve the shadowing problem in every configuration. It handles self-shadowing and does not need any additional information such as a separation between occluders and receivers. Lastly, the resulting shadows have an image-based definition. But it has three major disadvantages. Firstly it computes only hard shadows. Then the rendering process becomes quite slow due to the additional rendering needed. Finally, it is not able to handle the cases where the camera is in a shadow volume. This last problem is an important issue. In fact, this problematic configuration occurs often. For example, in our case, the major part of the scene is in the shadow of the external light source (the sun).

The reason of this mishandling is easy to understand. We recall that the shadow volumes method is based on a count of the entered and left shadow volumes performed via a stencil buffer in which these volumes are rendered. When the view point or even just the near view plane of the camera is in one or more shadow volumes, some parts of them will be clipped. If the clipped polygons face front, the count will be inaccurate and will lead to false results in the stencil buffer and thus to a final misplacement of the shadows.

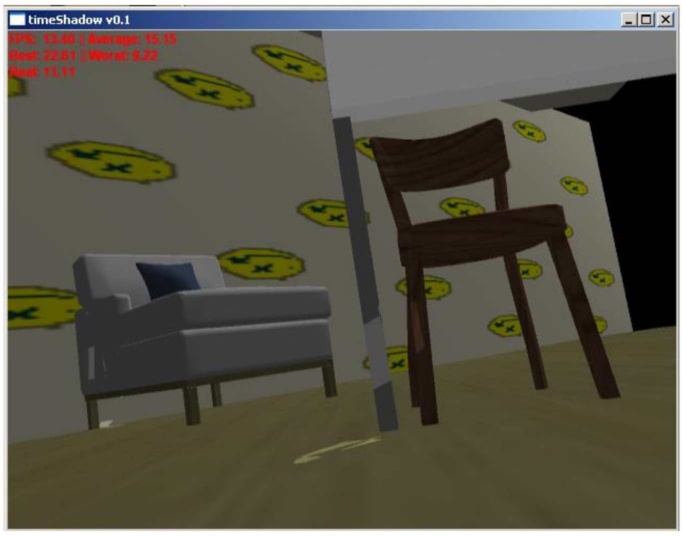


Figure 35 : False shadows when the camera enters the volumes

The most common method to fix this problem is the Zfail algorithm (by opposition to the previous one, described as Zpass). Both of these algorithms are based on the same principle but differ in two ways. In the Zpass method, we first render the shadow volumes front faces and then the back faces. In the Zfail technique, the rendering order is inverted. Moreover, we need to use enclosed shadow volumes for this improved method. This means that we have to cap the volumes at both ends. We therefore need close capping that could be the shadowed part of the considered geometry and a far cap. As we can see, a lot of additional geometries are needed and that will increase the fill need, which is the major bottleneck of this algorithm. Moreover the used geometries can have a bad connectivity and we could have trouble to actually close the volumes. Because of these problems, we finally chose to implement another method to solve this problem: the ZP+ algorithm.

ZP+ stands for "correct Zpass". This solution was developed by S. Hornus et al. [18]. It is a robust algorithm based on the Zpass method that solves the problems stated earlier. Instead of performing the shadow volumes count from the camera point of view starting with an empty stencil buffer, we will start by initializing this buffer correctly. This first step aims to take into account the misclipped volumes. In order to perform this initialisation, the authors proposed to build a view frustum that contains the problematic geometries. The view point of this view frustum will be the light location. Then, using an off-center projection, we will set its far view plane to the camera near view plane. Thus, the problematic shadow volumes that are the ones between the light and the camera near view plane will be contained in this new view frustum. Finally, we only need an additional rendering step to initialise the stencil buffer. In fact, we first render the scene shadow volumes according to the classic Zpass algorithm but from the light point of view and in the previously described view frustum. Another change during this first pass is due to the fact that we need close-capped shadow volumes. At the end of this additional step, the stencil buffer will be correctly initialised with the previously clipped volumes and we will be able to perform the classic Zpass method from the camera point of view and to get a correct stencil count and then a true shadow placement.

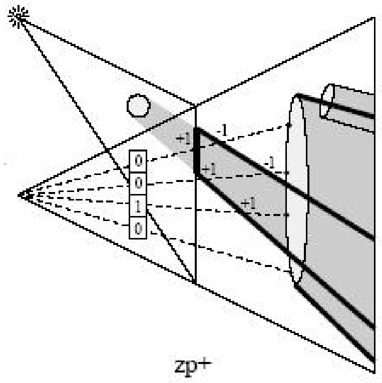


Figure 36 : The additional view-frustum

The final algorithm to compute the needed stencil buffer for a light is:

```
Clear the z-buffer, the stencil buffer and the back buffer
Disable colour writing, z writing and z testing
Enable stencil buffer writing
Set the view point at the light position
Set the view frustum with a far plane equal to the camera near plane
Render the front faces of the near-capped shadow volumes by increasing the stencil
    count
Render the back faces of the near-capped shadow volumes by decreasing the stencil
    count
Set the view point to camera position
Set the view frustum to the camera one
Enable colour writing, z testing and z writing
Disable stencil buffer writing
Render the scene to initialise the z-buffer
Disable z writing
Enable stencil buffer writing
Render front faces of the shadow volumes by increasing the stencil count
Render back faces of the shadow volumes by decreasing the stencil count
```

Algorithm 10 : The ZP+ algorithm

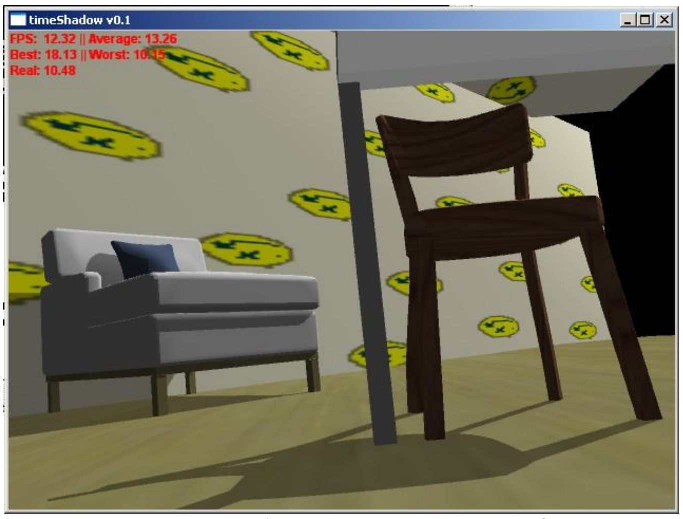


Figure 37 : The shadow count is corrected by the use of the ZP+ algorithm

ii. An optimisation

Although this solution gives good results, it is quite time consuming. Indeed, it requires an additional shadow volumes rendering pass. Furthermore, during this first step, we have to render close capped shadow volumes that are complex geometries. This additional rendering pass is the main cause of the time cost of ZP+. Thus we aim at optimizing this improved method. First of all, it would make sense to use it only when needed, i.e. only when the problematic configuration with mis-clipping occurs. Then, by limiting the number of rendered shadow volumes during this initialisation, major time gain could be obtained.

As mentioned before, the classical Zpass algorithm fails when some sides of the shadow volumes are missed. This can happen when the camera is in a shadow volume or when the near view plane clips shadow volumes sides. Indeed, we can combine these two cases by using a small pyramid with the camera at its top and the camera near view plane for basis. Thus the two previous cases can be summed up by saying that problems in the Zpass algorithm occur when this pyramid intersects one or more shadow volumes. And we just need to render during the first step these intersected shadow volumes. The determination of the possible intersection between the pyramid and the complex geometries of the shadow volumes is a difficult problem. We can make some approximations to test this intersection more quickly. In fact, we will study the intersection between the sphere containing the "camera pyramid" and the shadow volumes extruded from the bounding spheres of the geometries. This approximation is rough but is cheap to compute. Moreover, for the same reasons as for our shadows culling, this test can lead us to consider too many shadow volumes (but never too few).

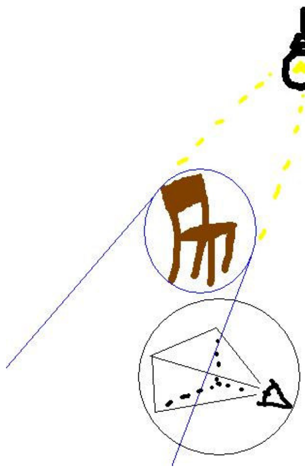


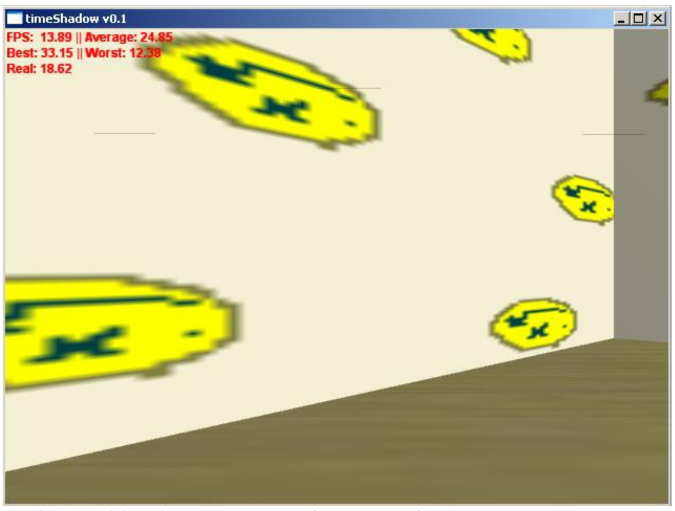
Figure 38 : The principle of our optimisation. The sphere including the camera view frustum is tested against the shadow volumes projected by the objects bounding spheres

Thus before the ZP+ step of the shadow volumes algorithm, we test all objects to determine if their shadow volumes intersect the camera pyramid and build a list of the ones who do. If this list stays empty, we clean the stencil buffer and jump directly to the main part where shadow volumes are rendered from the camera point of view. Otherwise, we initialise the stencil buffer by rendering the listed shadow volumes from the light point of view before performing the second part of the algorithm. This testing allows us to avoid the use of the ZP+ in most cases and improve its speed by rendering only the needed geometries.

iii. Results

Thanks to this improvement, we avoid the frame rate drop in most cases because of the ZP+ step. Moreover, reducing the number of objects involved during the ZP+ stencil buffer initialisation keeps the rendering speed high. Thus, we obtain frame rates around 40fps for a single light on slower computer when the camera stays outside of all shadow volumes. When the camera enters a volume, the frame rate decreases but stays above 20fps.

Nevertheless, the ZP+ algorithm has two major disadvantages. First of all, some artefacts cannot be avoided. Indeed, on the rendered pictures, there are some pixels for which the stencil count seems to stay false. This kind of error was reported by the authors of the algorithm. They explained it by numerical error during the graphical computations. In fact, the two rendering steps of the shadow volumes performed during the ZP+ process are done in two different view frustums. This means that their vertices are projected and clipped using two different projection matrixes. Even if these arrays theoretically match perfectly, the values in the matrices are rounded off. This can lead to rounding errors and prevent perfect matching between the volumes rendered in the two view frustums. This is how the clipping errors between the far view plane of the light view frustum and the near view plane of the camera can lead to these artefacts.



```
FPS: 13.89 || Average: 24.85
Best: 33.15 || Worst: 12.38
Real: 18.62
```

Figure 39 : An example of the artefacts that can appear

The second disadvantage of this method is that it is not able to handle a very special case. When the light source is in the camera near view plan, the camera view frustum used during the first rendering step of the ZP+ will be totally flat and hence empty. This volume was used to perform a rendering that initializes the stencil buffer count. Because the view frustum is flat, this initialisation becomes impossible and the ZP+ method does not give a correct count of the shadows.

But these two problems are not so crucial. First, the artefacts are caused by errors in small buffer areas. Thus, in the worst of the studied cases, we only noticed small lines where the shadowing was incorrect. Moreover they do not occur very often and most of the pictures are correct. Furthermore, as we will describe in the next part, we decided to compute the shadowing of the biggest receivers by using shadow maps instead of shadow volumes. This means that the results of the stencil buffer will not be taken into account for most pixels. Thus the artefacts will not occur where shadow maps are used and there will be even fewer remaining false shadows. Finally, the configuration which is not handled is a very special case. In fact, it is impossible for the light to lie exactly on the camera view plan. During the tests, this situation never occurred. So we chose to leave this problem unsolved.

b. *Real lighting: the original shadow volume algorithm*

The previously implemented shadow volumes algorithm is not the original method. The first step is the same in both cases: we render the shadow volumes in the stencil buffer to perform the shadow count. The results in the stencil buffer enable us to determine the shadowed pixels. They are the ones where the stencil count is positive. In our implemented method, we render a transparent grey square where the shadows darken the scene. This technique is interesting because it spares us a scene rendering. But it has two disadvantages. First, the back facing parts of the objects are shadowed twice: first by the shading model and then by the rendering of the grey square. Moreover, it approximates the shadow colours. This is not so problematic for a single light, but will produce very inaccurate results if several light sources are present. This led us to implement the real shadow volume algorithm that is more time consuming but which allows us to support multi-lighting and solves the double shadowing issue.

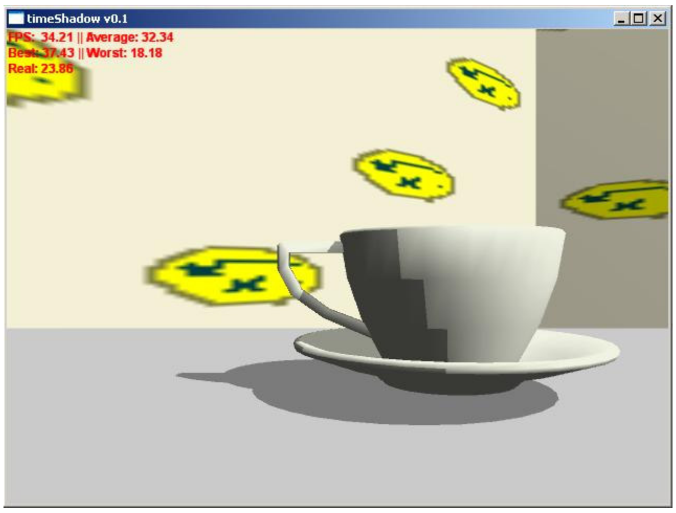


Figure 40 : The double shadowing issue is here visible on the cup: a part of it is too dark

In this method, we first perform an ambient pass. During this first step, we render the scene only with the ambient light and with the classical rendering state in order to initialise the back buffer and the z-buffer. Then, for each light, we compute the stencil buffer resulting from the shadow volumes of the current light source. This step can be done with or without the use of the ZP+. Finally, we switch off the ambient light and turn on the current one. We perform then a lit pass during which the scene is rendered and added to the previous back buffer only where the shadow count is negative or null i.e. only where the objects are actually lit by the light source. The classical algorithm is:

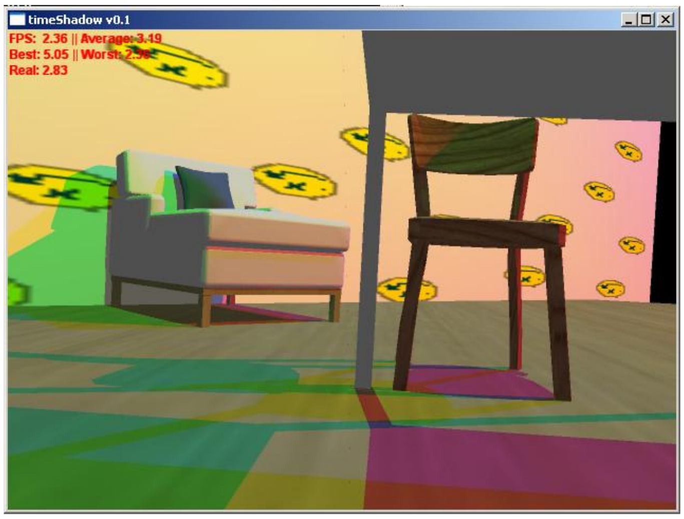
```
Clear the back- and z buffers
Enable colour writing, z-writing and z-testing
Turn the lights off
Turn the ambient light on
Render the scene
Turn the ambient light off
For each light li:
  Initialise the stencil buffer with li
  Switch on li
Render the scene where the stencil count is not positive by adding the result to
    the back buffer
Switch off li
Next li
Show the back buffer
```

Algorithm 11 : The original shadow volumes algorithm

According to our expectations, the real shadow volumes method allows us to get pictures with more realistic shadows. In fact, this process approximates the light equation fairly accurately:

$$I_{total} = texture \times I_{ambient} + \sum_{lights} \delta_i \times texture \times I_i ,$$

where I_{total} is the resulting pixel colour, *texture* its texture value, $I_{ambient}$ the light intensity due to the ambient light and I_i the light intensity at this pixel received from the light i. δ_i indicates if the pixel is in the shadow or not. It has to be set to 1 if the pixel is lit by the light I_i and to 0 otherwise.



```
FPS:  2.36 || Average: 3.19
Best: 5.05 || Worst: 2.36
Real: 2.83
```

Figure 41 : Shadow volumes with multi-lighting and coloured light sources

Nevertheless, by adding an additional scene rendering pass instead of a simple square rendering, the original algorithm is more time-consuming. Indeed, to render the shadows of a scene lit by N lights, N+1 renderings will be performed. We therefore chose to keep the possibility to switch between these two solutions in order to get an extended rendering speed range.

c. *Shadow volumes cropping*

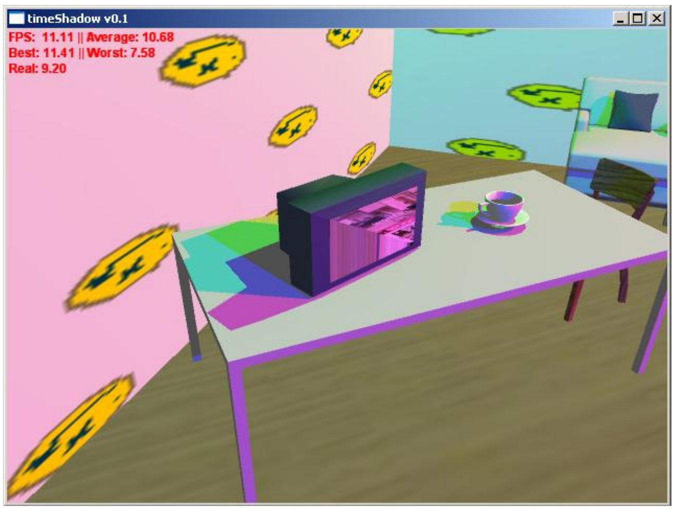
Until now, we approached the shadowing problem in two ways. Either we implemented shadow volumes to handle every kind of object configuration and produce hard shadows at image definition. Or we used the projected texture shadows technique and its improvements. This method provided us with shadows that could be physically exact but it required a strict distinction between the planar receivers and the occluders. This technique seems to be well adapted to our case where we actually have well identified large receivers. Thus we could get exact soft shadows on those for a small rendering cost. Nevertheless, some objects occlude themselves and a lot of shadows are cast on scene objects by other ones. The shadow volumes method is the only one that is able to solve this problem, so it could be interesting to combine the two solutions. In fact, the occluders are generally quite small. Thus the shadows cast on them could be computed as hard even if they are a little bit soft like all real world shadows. By using shadow map for the extended objects, we will get nice soft shadows when required.

To be able to let the two shadowing possibilities run together it is essential to be able to render some scene objects (e.g. the pure planar receivers), without the shadow volumes influence. It can be easily done by splitting the rendering stage into two parts. First, we render the occluders with a classic shadow volumes rendering. Then we disable the stencil buffer and render the receivers. Thus the receivers will not be affected by the result of the stencil buffer and so will not be shadowed. Moreover, the artefacts due to the ZP+ will mostly disappear as explained previously. The principle is described bellow:

```
Clear the back- and z-buffer.
Render into the back buffer the shadowed occluders using the shadow volume
    algorithm.
Disable the stencil buffer.
Render the receivers.
Show the back buffer.
```

Algorithm 12 : The shadow volumes cropping algorithm

Thus we get the desired results at a frame rate that is around that of the classic shadow volumes. In fact the additional costs of splitting the rendering process are compensated by the fact that the receivers do not take part to the multi-pass rendering of the shadow volumes algorithm and are rendered only once.



```
FPS: 11.11 || Average: 10.68
Best: 11.41 || Worst: 7.58
Real: 9.20
```

Figure 42 : The scene rendered with cropped shadow volumes

3. The final rendering process

a. *Analysis*

i. *The shadow maps algorithms*

We implemented three improvements of the projected texture shadows algorithm. The following table contains the final frame rates for the motionless rendering of the test scene shown in the part V.1. for each of the four algorithms. The testing framework includes a single coloured light. The shadow maps are generated for the four receivers (the three walls and the floor) at each frame. We call Computer 1 the slower one and Computer 2 the faster one.

Tested computer	Computer 1	Computer 2
Hard projected texture shadows	66.0	218.0
Blurred projected texture shadows	14.2	34.0
Sampled projected texture shadows (with 20 samples)	4.4	9.7
Smoothies algorithm	10.5	32.4

Table 2 : The frame rate measured for the several shadow mapping modes

We have first to note that the evolution of the frame rates for the several methods follows the same scheme on both computers. Thus the conclusions that we will get from the analysis of these numbers will stay correct whatever the computer speed will be.

The sampled shadows techniques stays very expensive even on the quickest computer. But it does not signify that we will not be able to use this algorithm for our real-time rendering. Indeed we have to remember that the shadow maps will mostly stay the same during several consecutive frames. Then, when they will be computed, even if it costs a lot of computation time, we will be able to reuse them without any calculation. Thus we point out here that even the slowest computer achieves to compute more than 16 shadow maps every second.

One of the implemented methods provides hard shadow maps very quickly and seems to be totally adapted to our problematic. Another one, the sampled shadow maps algorithm, handles physically exact shadows but will not be suited to handle dynamically the changes in the scene. Finally, we have two algorithms that produce fake soft shadows at a reasonable frame rate. On the first hand, the blurred projected shadow maps technique provides totally false approximations of the shadows. On the other hand, the results of the smoothies approach are also fake but the approximation, even if it is rough, gives attractive results. Since their measured frame rates are close, we chose to abandon the use of the blurring process. Moreover, the costs of the smoothies are divided in two steps. During the first tests, geometrical computations on the objects are performed and during the second one, the built smoothies and their geometries are rendered. Contrary to the blurred maps, when an object changes between two frames, only its smoothies and the rendering step have to be updated and an important part of the recreation process will be spared, which is why this algorithm seems to be the more suited to the quick generation of fake soft shadows.

ii. The shadow volumes algorithms

Two basic improvements of the firstly implemented shadow volumes method were implemented: the original algorithm that supports real lighting and the possibility to exclude the receivers of this shadowing process. Once again we performed the measures with a single light, without using the ZP+ modification and by re-computing all the needed data (here the shadow volumes geometries) at each frame.

Tested computer	Computer 1	Computer 2
Without improvements	35.9	120.0
With cropped shadow volumes	32.2	115.0
With correct lighting	29.8	103.4
With the two improvements	28.9	97.0

Table 3 : The frame rates measured with the different shadow volumes modes. The rendered scene is our test scene.

The bottleneck of the shadow volume algorithms is the fill rate. Thus the general frame rate stays proportional to the number of shadow volumes polygons. In a scene with N lights, the number of rendered volumes will be N times higher and the frame rate roughly N times lower. Thus we see that the handling of several lights on the slowest computer will lead us to a rendering speed that will stay higher than the minimum needed for an interactive rendering. With a faster hardware setup, shadow volumes will stay handled in real time even with multi-lighting. Moreover, most of the shadow volumes will not have to be calculated at each loop and we will achieve interesting frame rate gains.

Furthermore, these measures confirm that the real-lighting algorithm is a little bit more expensive as well as the version that we called "cropped shadow volumes". In spite of this small frame drop, this second improvement does not damage so much the overall performance and will be useable in all the cases. The frame rate improvement due to the use of the false lit method is enough advantageous to keep using this method on slowest computers or for a quickly changing environment we will keep using the false lit method.

We modified the basic shadow volumes algorithm by including the ZP+ process. This major change leads to some modifications in the observed frame rates. The following results were measured during the rendering of our test scene with one light and on the one hand the basic shadow volumes algorithm and on the other hand the same method with the two previous improvements. We have to distinguish two cases. Firstly when the camera is outside all the volumes and that our optimisation of the ZP+ switches it off and then when the point of view stays in an occluded part and that the ZP+ pre-computations are actually done.

Camera location and tested computer	Outside the shadows Computer 1	Outside the shadows Computer 2	Inside the shadows Computer 1	Inside the shadows Computer 2
Without the improvements	34.6	118.0	16.6	68.9
With the two improvements	30.0	95.5	15.4	60.7

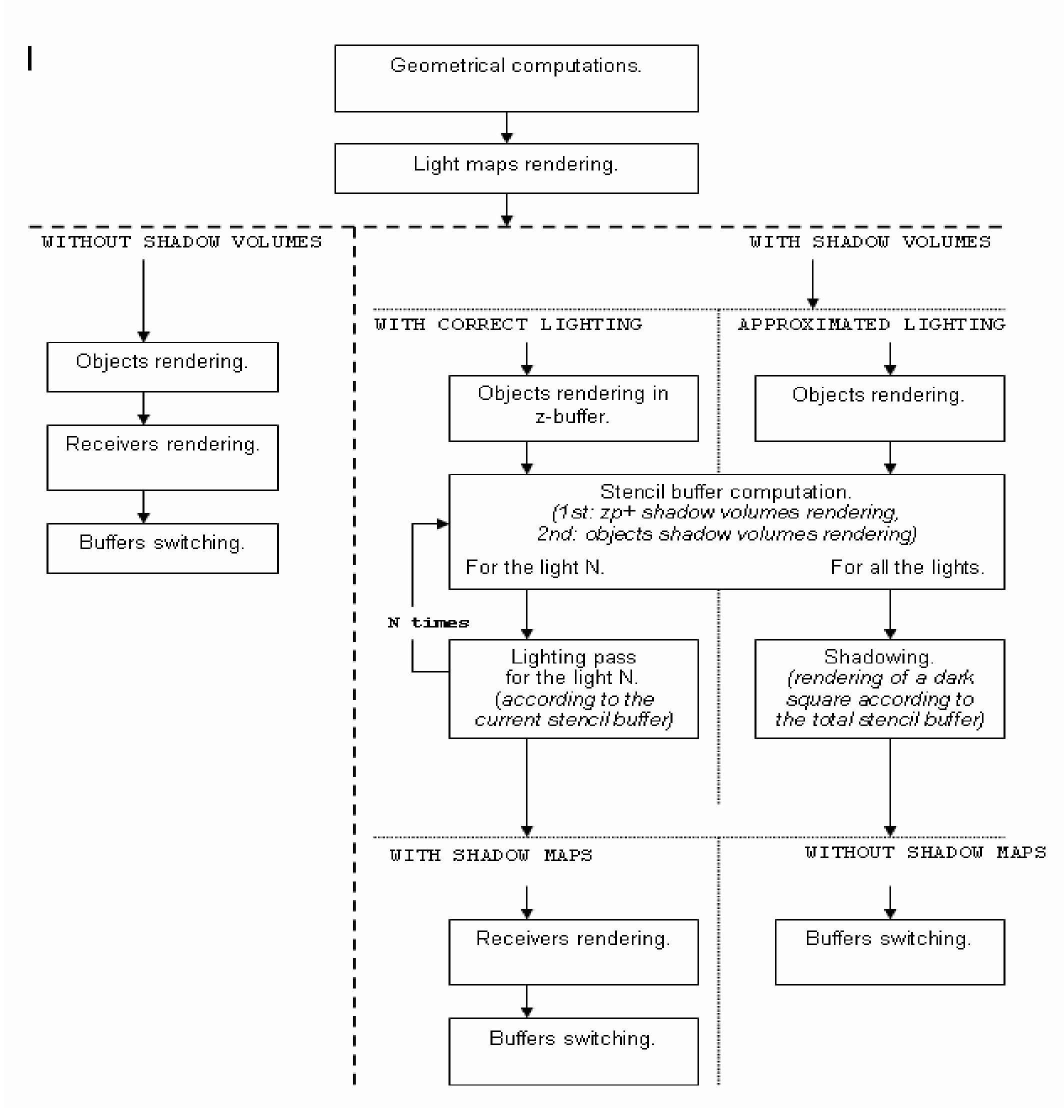
Table 4 : Study of the frame rates performed with the ZP+ algorithm

These results first show that the needed additional computations for our optimisation of the ZP+ algorithm do not influence so much the results. Thus, the determination of the position of the camera relatively to the shadows volume does not reduce the rendering rate. Moreover, we see that this test part is very useful because the frame rate decreases dramatically when entering a shadow volume. But even with this major reduction we get interesting results that stay in the real-time area. The ZP+ allows us the shadowing of a scene for all the camera positions, which is indispensable. Thus we will use it in any situation. Moreover, in our study, the viewer position stays mostly outside the shadows that are cast by the furniture towards the floor. So the ZP+ step will be skipped in most cases.

b. *The rendering pipeline*

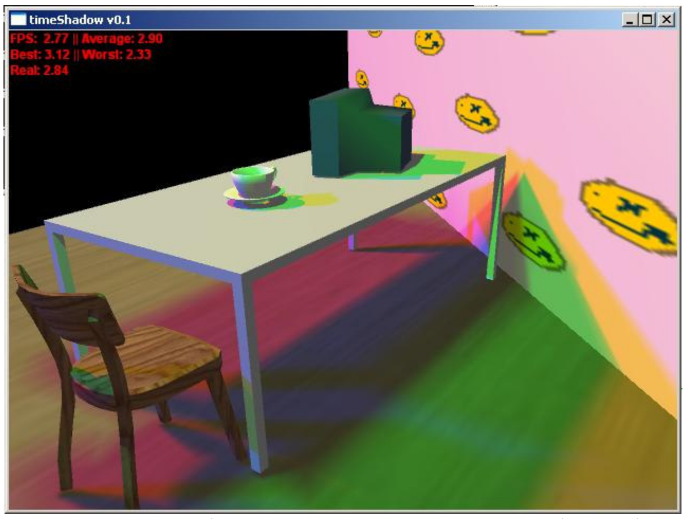
As written before, we modified the shadow volumes algorithm to be able to render some of objects without shadowing. We have done this modification with the intention to render the receivers with their shadow maps. Thus we modified the rendering loop to let it run in a single pipeline.

This new process is divided into two main steps. During the first part, the needed data is computed. It concerns the geometrical constructions (shadow volumes, smoothies and the listing of the involved objects for the ZP+ pre-rendering) and the rendering of the shadow maps according to the chosen generation method. Then we actually rendered the scene according to some parameters. These parameters determine if we use shadow volumes, the lighting model for them if we do and if we are to use shadow maps. They are determined by the chosen rendering parameters and by hardware capabilities. The rendering algorithm is then:



Graph 3 : The final rendering pipeline

This new rendering process allows us to get pictures where self-shadowing and complex configurations are handled and where the biggest shadows have a higher quality thanks to the shadow mapping. Moreover, it is flexible and easily tuneable to adapt the rendering speed to the computer's hardware capabilities.



```
FPS:  2.77 || Average: 2.90
Best: 3.12 || Worst: 2.33
Real: 2.84
```

Figure 43 : The scene rendered with sampled shadow map combined with shadow volumes

VII. The real-time issue

Now that we have developed a rendering process that matches our expectations from a quality point of view, we have to make it run in real time on every computer. This part is focused on the modifications that we did to be able to tune the frame rate. We will first study the handling of the scene staticity, then the modifications implemented in the pre-processing parts and finally the determination of the rendering parameters. This is the final algorithm that produces high-quality shadowing in real time and adapts itself to the hardware capabilities.

1. Staticity handling

a. The principle

Until now, all the data we needed (geometrical constructions and the shadow maps) was computed at each frame during the pre-processing step. Since the scene is mostly static, there are often no changes at all or just a few between two frames. The changes result mainly from the user interactions through the drag-and-drop interface. Thus the differences mainly come from the displacement of an object and of its models, from its resizing, from the change of the characteristic of a light, from the suppression or the addition of a few geometries. Nevertheless, the scene can change completely from one rendering to the next so we have to support it but it is far from being the most common case. The computation step is a very expensive part of our pipeline. Thus it could be interesting to reduce it as much as possible in order to get major frame rate improvements.

Instead of massively re-computing the data at the beginning of the rendering loop, we will first test the validity of the existing ones and calculate them only when needed. If neither the objects nor the lights positions change, the shadow volumes stay relevant. Similarly, the pre-smoothies (first calculation step of the smoothies algorithm) can be re-used in this case. The smoothies in themselves need valid pre-smoothies and an unchanged receiver geometry. Finally, the light maps have to be recomputed if any of the casting occluders change, if the receiver geometry evolves or if the light position or lighting characteristics are modified. Therefore, we add a first stage where the data relevance is tested to our rendering pipeline. Only the ones that have to be updated will be sent to the computation step.

In order to test this principle we modified the rendering scheme by skipping the computation step after the first loop. Indeed, we are rendering a static scene where no changes occur. Thus, we use the data calculated during the very first frame rendering for each frames. The following frame rates were measured with three lights, each of them casting shadow volumes. The receivers were rendered with sampled projected texture shadows.

	With real lighting for the shadow volumes		With approximated lighting for the shadow volumes	
	Computer 1	Computer 2	Computer 1	Computer 2
Resulting frame rate	38	128	50	186

Table 5 : Frame rates for a static scene

The observed frame rates are very promising. Even on the slowest computer, they stay higher than the real-time requirements. But two aspects have to be considered. On the one hand, our test scene is very simple and we will have to deal with more complex scenes where the number and the complexity of the geometries will be higher. Moreover, the number of simultaneous involved lights can be up to five in the Navigram software. Thus, the increasing scene complexity will give us much lower rendering rates. On the other hand the scenes are not totally static so we will have to perform additional computations at the beginning of each rendering loop in order to determine the missing data. Furthermore, the engine is controlled by an application that will need CPU time which it will have to share with other programs, so the hardware capabilities will not be completely available.

Because of these restrictions and despite the good results, we will surely not be able to always perform a real-time rendering with the highest shadowing quality. This is why we will have to work on a kind of optimiser that will allow the rendering process to automatically adapt itself to adjust the performances to the computer's abilities.

b. Temporary objects handling

The hardest part of this adaptation will be how the algorithm handles non-permanent objects. A typical example of a temporary object is a dragged item. Before it is selected, the model stays unchanged and will be called permanent. But as soon as it begins to be moved, its geometry changes at every frame until the user releases it. This kind of behaviour leads us to make the distinction between so-called permanent objects that will probably stay unchanged until the next frame and temporary geometries that will change in the following rendering loop.

We studied three different ways to handle temporary objects.

(1) The first possibility was to handle them exactly the same way as permanent ones. We studied two situations. In the first situation, all the objects have shadow volumes and cast sampled shadows on the receiver: these are the *"with sampled temp. maps"* measures. In the second situation, the *"with smoothies temp. maps"* sample, we use the same rendering pipeline but compute the shadow maps with the smoothies approach instead of using the sampled version.

(2) The second possibility is the *"with shadow volumes and without maps"*. Here, we only used shadow volumes and used this algorithm to generate the shadows cast on the receivers as well.

(3) The last possibility results from another approach. When an object begins to change, we calculate the shadow maps of the receivers without taking the temporary objects into account. Thus, only permanent geometries are shadowing on the used light textures. This rendering possibility is shown with the measures *"with sampled permanent maps"*. To let the dragged objects cast shadows, which is very important to understand their positions during their movement, we will use a separate shadow volume pass only for them. This process is studied with the *"with sampled permanent maps and temp. shadow volumes"*.

We performed the presented test by moving the chair of our scene from the third second to the sixth second. The measures were made on the slowest computer.

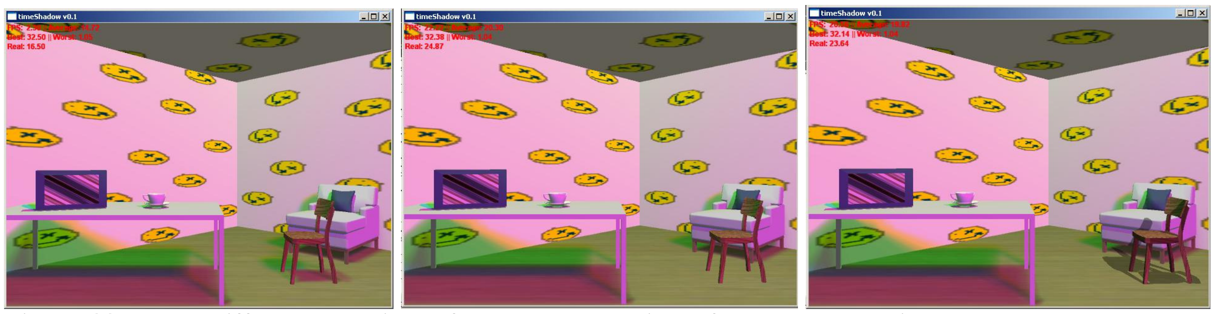
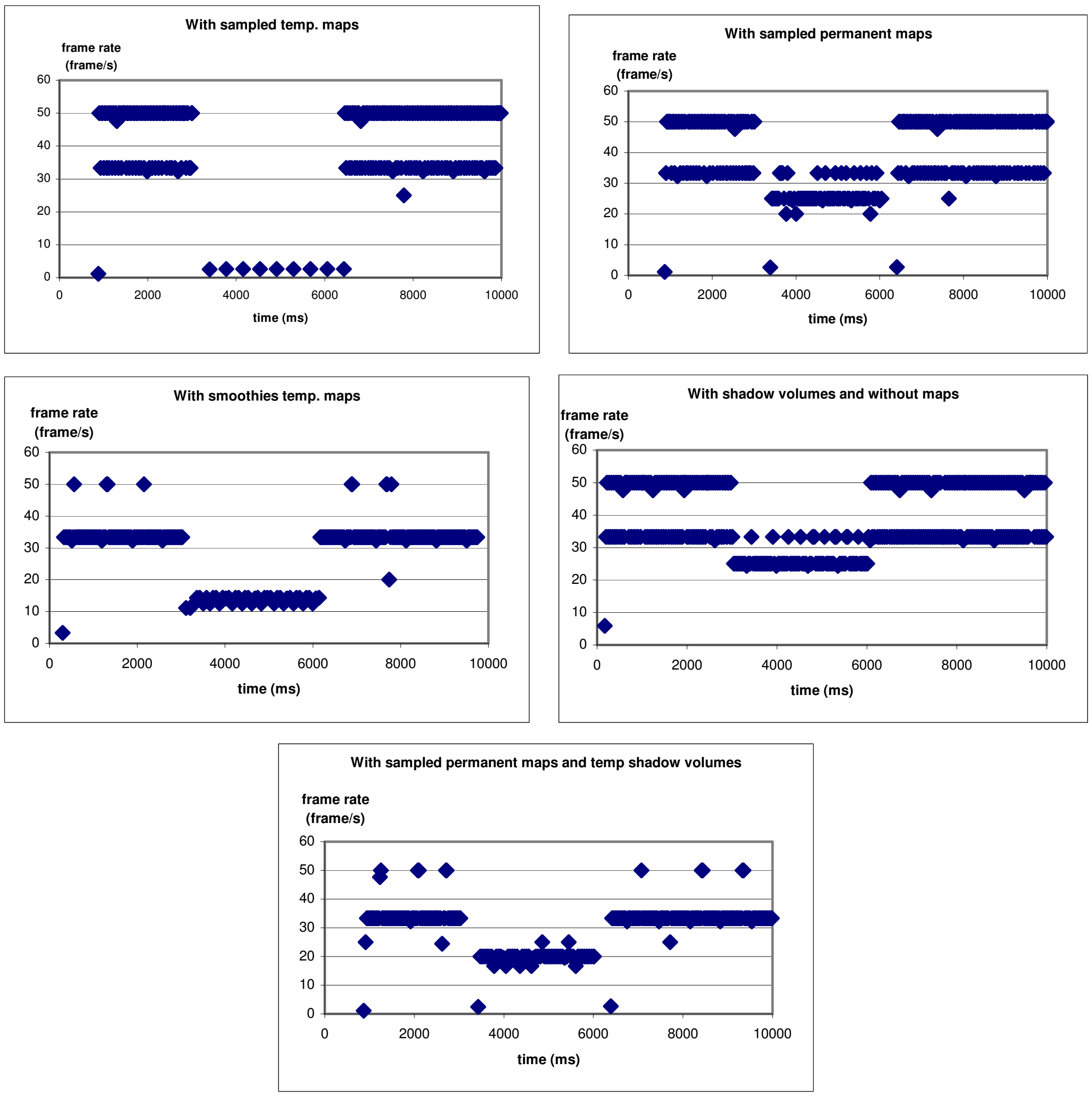


Figure 44 : Three different handlings of the dragged chair. Left: temporary objects cast the same shadows as the permanent ones. Center: Only the permanent ones cast shadows. Right: The permanent objects are taken into account to compute the shadow maps and an additional shadow volumes pass is used to get the shadow of the temporary geometry.



Graph 4 : Measured frame rates when moving a chair

The rendering speed for static scenes stays around 40 fps, which is comparable to the frame rates presented in the previous part. As expected, the computer is not able to generate new sampled light maps in real time. During this test, the rendering frame rate stayed around 2fps. This is insufficient, even for an interactive rendering. The four other methods achieved to render at least at interactive frame rates (for the method with smoothies maps regeneration at each frame) and for the three best ones in real time. The fastest approach is the one that does not use shadow maps. In our sample case, the computer achieved to generate new shadow volumes quickly enough for the rendering to be always shadowed and delays between frames always inferior to 50ms. But the final image quality is not as good as the results provided by the mapping of light maps on the receivers. Thus, the methods that keep rendering with shadow maps are better, even if they are a little bit slower. The regeneration of shadow maps with the smoothies algorithm is adapted to our case but does not have enough progression margin to be a solution for more complex scenes. So the two more reliable methods are the ones that use shadow maps computed only with permanent objects. By avoiding this additional processing, we get very good frame rates and if we add an extra shadow volumes pass we are able to keep providing spatial information to the viewer.

We have come to the conclusion that we have on the one hand an algorithm able to render high quality shadows at low frame rates but that could be efficient on fast computers for some scenes. And on the other hand, we implemented a process that computes in real time very good results that fulfils our expectations about quality and information supplied by the shadowing. Between these two opposites we have a range of possibilities that will allow us to adapt closely the rendering pipeline to the computer capacities.

2. The shadow maps generation costs

a. The issue

In the previous part we presented two techniques that seemed able to render in real time. But it is not completely true. In fact, we can observe at the beginning of the chair movement that during one frame, the rendering speed decreases dramatically. This very slow rendering loop is the one during which we have to compute the new light maps. This issue is confirmed by the time needed for the very first frame. For the two algorithms that use sampled shadow maps, we need up to one second to compute these maps and to render the scene for the first time. These major frame drops during the generation of the shadowing textures are preoccupying. Indeed when we resize the rendering window, because of the DirectX API, we have to clear the RENDER_TARGET textures and so our shadow maps. Thus, the generation of shadow maps will always be needed and we cannot provide long-lasting frozen pictures. Therefore, we proceed to some tests to measure the needed generation time of these maps. The results given bellow are the times needed by the several algorithms to generate the light map of the floor of our final scene. For the smoothies method, the computing of the smoothies' geometries was not included.

Computer	Computer 1	Computer 2
Hard	30ms	20ms
Smoothies	30ms	20ms
Sampled	1020ms	250ms

Table 6 : Shadow maps generation times.

We can firstly notice that the generation of a simple light map with the hard algorithm or with the smoothies method stays possible in real time. Both techniques have a similar rendering cost. This means that as soon as we already have computed the smoothies, this algorithm that produce nice fake shadows with a single rendering will be used. But they will not be able to generate the several light maps needed for a scene and to handle the shadowing produced by more complex occluders geometries enough quickly. If we wait to compute all the shadow maps of a scene before to render it, we will often observed frozen images due to a punctual but noticeable frame drop. This issue is even worse for the sampled algorithm and its huge cost.

b. The implemented solution

We introduce a first modification to our rendering pipeline to support these major computation costs. First of all, the available time for the pre-processing step will be restricted to respect a minimal frame rate and avoid the bottleneck due to this stage. According to the length of the previous rendering, we will determine this time. During this fixed period we will compute as much shadow maps as possible. Then, we will determine at the end of this generation step if all the needed shadow maps are available. If they are, we will execute the pipeline with shadow mapping for the receiver. Otherwise the receiver will be rendered as the other objects. Finally we add an improvement to take advantage of the rapidity of the hard shadows algorithm and of the quality of the samples map. We will firstly compute during a few frames the hard version of the textures and then use it. At the same time we will perform the computation of the more expensive maps and use it as soon as they will be available. Thus, our algorithm will use shadow maps very quickly (at every frame on quickest computers) and will continue to improve their quality meanwhile.

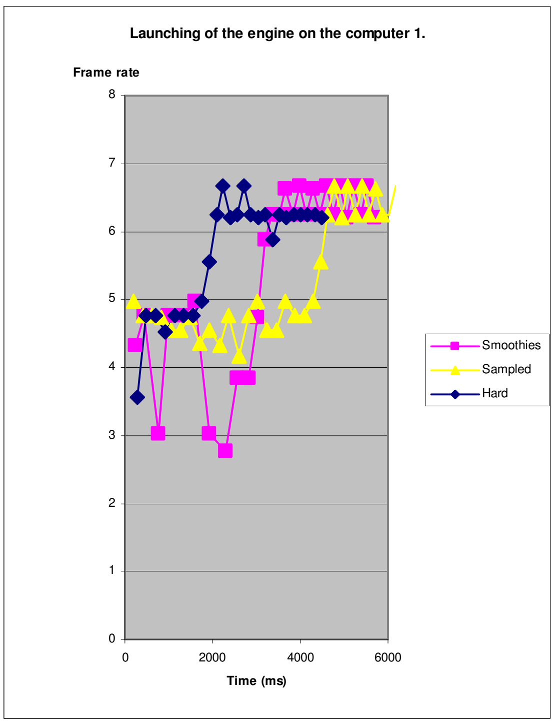
Thanks to this improvement we get less frozen pictures when shadow maps computations are needed. But the issue stays unsolved. Thus, with a generation time that can be up to 30ms, the generation of hard or fake shadow maps stay too long to insure a stable rendering frame rate. Moreover, the cost of a sampled shadow map stays out of reach of a computation between two successive frames. For these reasons, we had to implement a second improvement to the generation process. We add the possibility to stop the calculation of a light map during the process and the ability to resume it. So we will be more able to finely constraint the time needed by the computation step. The rendering frame rate becomes more easily to adjust. We first split the several lighting passes. This means that the contributions of only some lights to the shadow map can be computed and that we are able to add the shadows produced by the other ones later. Thanks to this possibility, the generation of the light maps of our sample case can be done in three passes. Therefore, the minimal time required for the hard maps and the smoothies ones becomes around 10ms and that will allow us a fine control of the computation length. But even on the quickest computer, the length of the sampled version stays too prohibitive. Indeed we aim at rendering with frame rates that could stay very close to a desired value. The use of basic processes that could require up to 80ms cannot allow us to reduce the frame rate variation. Thus, we had to implement a finer tuning possibility for the sampled map generation. We split once again the generation algorithm but this time at the sample level. So the computations can be stopped and then resumed between two occluders renderings for two samples of one light. By this way we get a minimal length of less than 10ms that seems to be enough small to support an accurate tuning of the actual frame rate. Thus the implemented generation step follows this algorithm:

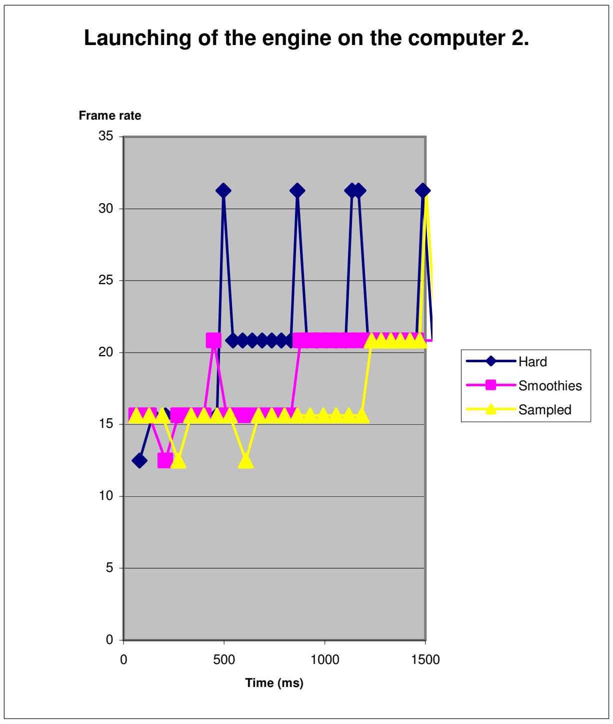
```
Determine the time available for computing
Update the shadow maps validity
While there is still time for computing:
  Select case:
    If there are still visible receivers without available hard shadows:
      Chose ri from one of these receivers
      Begin or resume the computing of the hard map of ri during the remaining time
      If the generation of the map is over, set it as up-to-date
    Break
    If there are still visible receivers without available smoothies shadows:
      Chose ri from one of these receivers
      Begin or resume the computing of the smoothies map of ri during the remaining
     time
      If the generation of the map is over set it as up-to-date
    Break
    If there are still visible receivers without available sampled shadows:
      Chose ri from one of these receivers
      Begin or resume the computing of the sampled map of ri during the remaining
     time
      If the generation of the map is over, set it as up-to-date
    Break
  EndSelectCase
EndWhile
If all the visible receivers have available shadow maps
  Execute the rendering pipeline with shadow mapping on the receivers by using
     their best map
Else
  Execute the rendering pipeline without shadow mapping
EndIf
```

Algorithm 13 : The optimised generation algorithm

c. Results and analysis

Using this improvement, we get the following frame rates at the start of our application when all shadow maps have to be computed. During these tests, we first compute the light maps and only use the shadow volumes when all the textures are available. On the slowest computer, we asked for an interactive frame rate of 5fps and on the quickest we aimed at 20fps. The rendered scene is the final one.



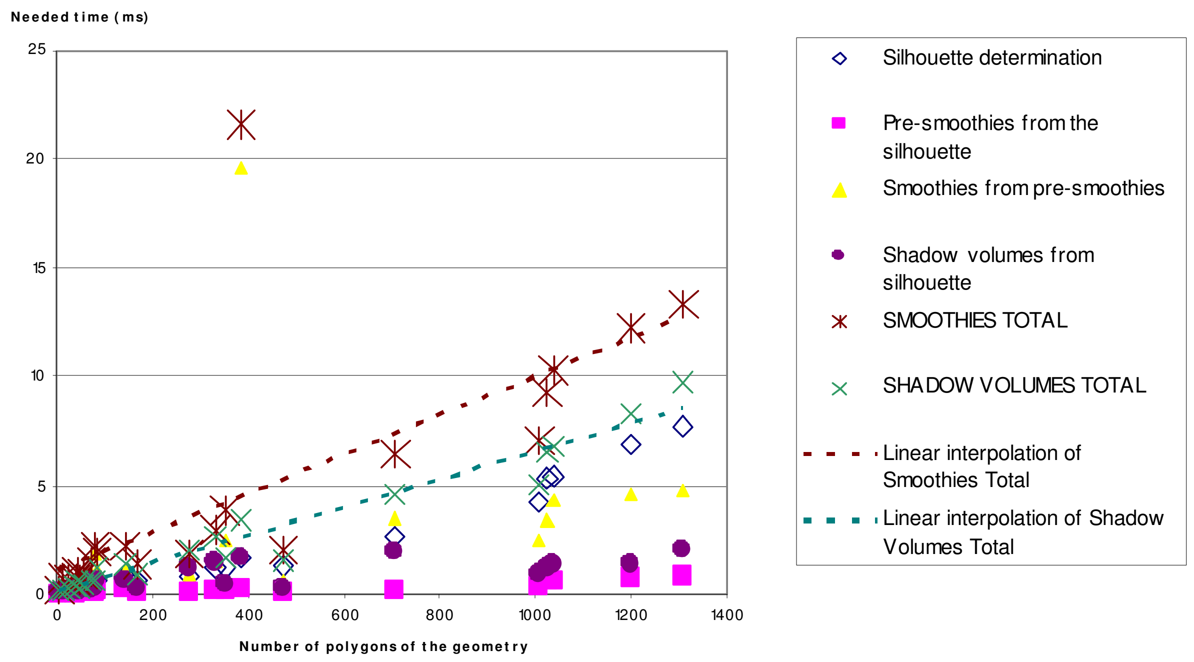


Graph 5 : Measured frame rates when starting the engine with the optimized maps generation process

The splitting of the generation process seems to be fruitful. Indeed, we manage to render scenes with a frame rate close to the required one, even when shadow maps are computed. This will allow us to avoid picture freezing during the starting or the resizing of the application and during the handling of temporary objects.

3. The computing costs

Even if it does not yet seem too problematic, we also have to consider the second part of the pre-processing: the computations of the geometrical constructions. Two processes are involved: the building of the smoothies and the determination of the shadow volumes. Both of them are based on the computing of the object silhouette. There are actually four basic algorithms (the determination of the silhouette, the extraction of the volumes from this edges list, the building on it of the pre-smoothies and the transformation of the pre-smoothies into smoothies). The different costs were measured on the slowest computer and are shown bellow:



Graph 6 : The measured computing costs

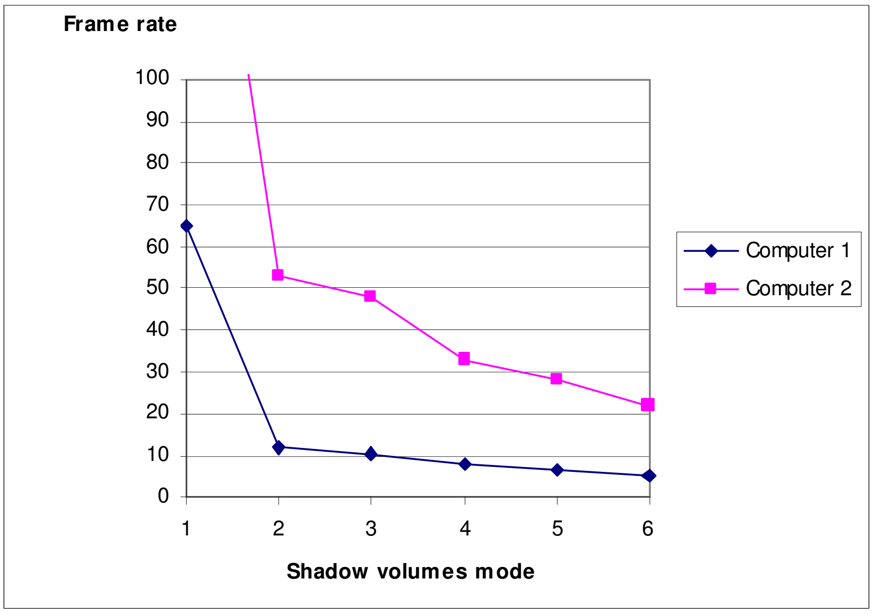
We observe that, even on the slowest computer and even for the more complex geometries, the cost of all these geometrical processes stays low (less than 10ms) but the computing of all the needed geometries cannot be done in real time between two frames, so we have to split it. This concerns only algorithmic manipulations and not graphical operations. It was then easy to do this processing in the available time by letting it run only on one object before to test if the computation can carry on. After this step, we used the shadow volumes shadowing only for the lights with all the needed volumes. This will determine the second part of the rendering parameters: the use or not of shadow volumes in the pipeline. Since we want to use shadow maps as much as possible for the receivers because they have a big influence on the quality of the picture, we decided to generate them first. Thus the smoothies have priority on the shadow volumes.

```
Determine the time available for computing
Resume the shadow map generation (cf. previous part)
While it remains time and unstudied light:
  Choose an unstudied light li
  While it remains time and unstudied receivers for the light li:
    Choose an unstudied receiver ri
    While it remains time and objects without smoothies for li and ri:
      Choose one of this object oi
      If oi has no silhouette for li:
        Compute the silhouette according to li
      EndIf
      If the silhouette has changed or oi has no pre-smoothies for li:
        Build the pre-smoothies based on the silhouette according to li
      EndIf
      Build the smoothies based on the pre-smoothies according to li-ri
    EndWhile
  EndWhile
  While it remains time and objects without shadow volumes for li:
    Choose on of this object oi
    Build the shadow volumes based on the silhouette according to li
  EndWhile
EndWhile
For each light li:
  If all objects have a shadow volume for li:
    Add li to the lights rendered with shadow volumes in the rendering parameters
  EndIf
EndIf
If there is at least one light in the rendering parameters:
  Execute the pipeline with shadow volumes for the determined lights
Else:
  Execute the pipeline without shadow volumes
EndIf
```

Algorithm 14 : The optimised computation algorithm

4. Considerations about the frame rate for shadow volumes

As previously discussed, the use or not of shadow mapping and the kind of algorithm used to generate the maps do not influence the rendering time since an additional texturing is quite cheap. Moreover, the differences between the several methods from a cost point of view appear in the pre-processing stage where we managed to regulate the length. Although shadow volumes require limited computations, it greatly affects the overall rendering speed. We tested this influence. The presented measures were done on the final scene by measuring only the rendering part length. The missing value on this graph is the rendering frame rate of the scene on the quickest computer and without the shadow volumes, it is more than 200fps.



Graph 7 : Measured frame rates according to the shadow volumes mode used. 1: Without shadow volumes. 2: With the shadow volumes casted by one approximate light. 3: With the shadow volumes casted by one real light. 4: With the shadow volumes casted by two real lights. 5: With the shadow volumes casted by three real lights. 6: With the shadow volumes casted by four real lights.

On this graph we can clearly see that the slowest computer is not able to let run any shadow volumes algorithm in real time. But, by handling only a restricted number of lights, it could run at interactive frame rates. On the other hand, the quickest computer can handle the five needed rendering pass in real time but will likely fail with a higher lights number in more complex scenes. Moreover, in order to render in real time we have to render quickly enough to produce the pictures at least at the required rhythm but we also have to perform computations at the same time. This means that the rendering in itself must leave enough available computing time for these calculations.

Another factor will affect the choice of the rendering parameters for the shadow volumes. We explained in the previous part that this determination was performed according to the available shadow volumes. The chosen shadow volumes mode must ensure, by interpolating the previous rendering speeds, that the rendering will be quick enough to respect the desired frame rate and to allow some computations if additional data is needed. As we can observe on the previous graph, a wide range of rendering costs is available and will give us the possibility to tune finely the length of the rendering part.

5. The final algorithm

The main steps of the final rendering process are now as follow. We call rendering parameters the set of choices that have to be done: number of lights handled by the shadow volumes algorithm, real lighting or approximate lighting for these lights and the used mode for the shadow maps.

```
Determine the maximal rendering parameters rp_fps according to the rendering speed
    in order to respect the frame rate and to allow computations
Determine the time available for computing
While it remains time to compute:
  Resume the shadow map generation (cf. the shadow map generation part)
  Resume the geometrical computations (cf. the computing costs part)
EndWhile
Determine the maximal rendering parameters rp_comp according to the available data
Determine the actually used rendering parameters rp_used according to rp_fps and
    rp_comp
Render the scene according to rp_used (cf. the rendering pipeline part)
```

Algorithm 15 : The final algorithm.

Thanks to this process we achieve to render the final scene with high-quality shadows at interactive frame rates on the slowest computer (around 6fps) and in real time on the quickest one (around 20fps). Moreover, the handling of the pre-computing step succeeds in avoiding any major frame drop between two frames. Even when shadow maps are computed the measured frame rate stays close to the desired one.

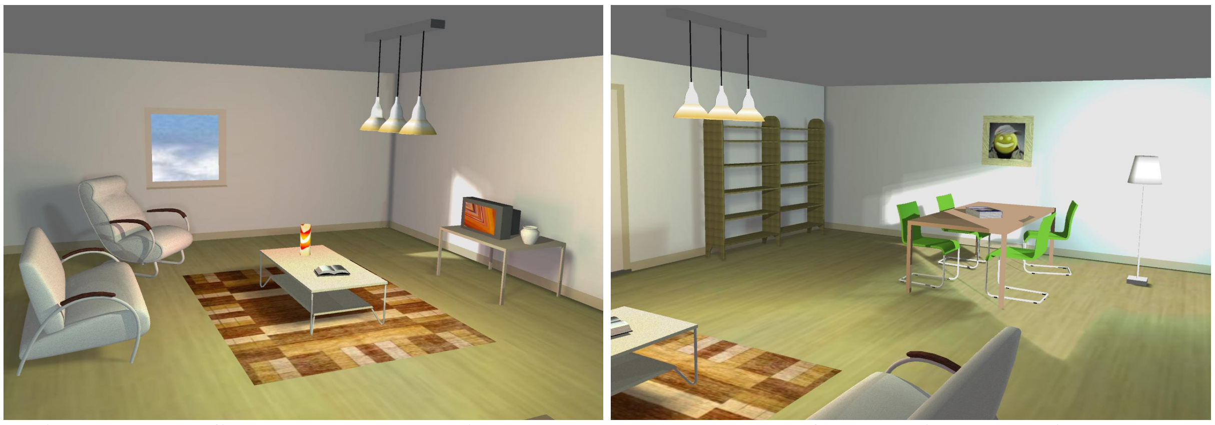


Figure 45 : The final scene rendered with shadow volumes casted by four real lights and with sampled shadow maps

When computations are performed, for example at the launching of the engine, we observe that the shadow maps quality improves along the rendering time. We firstly get pictures with un-shadowed receivers, then with hard shadows, then with smoothies generated textures and finally with the sampled version. Similarly, the number of lights handled by the shadow volumes algorithm increases continually until the maximal possibilities of the computer are reached. The improvement process is visible but is stable enough to avoid permanent switching. So it is not disruptive.

VIII. Conclusion

In this thesis, we presented our implementation of a rendering engine supporting soft-shadowing in real time even on slow computers for semi-static scenes.

We first discussed Navigram's requirements and found it was relevant to focus on static scenes with a few mobile objects. Then we briefly explained what shadowing is and did a short review of state-of-the-art solutions. The algorithms we presented are mainly based on two basic approaches: shadow maps and shadow volumes. Thus we implemented these two possibilities and added three different methods to get soft shadows: blurred shadow maps, sampled shadow maps and lastly an adaptation of the smoothies algorithm. To support the multi-lighting of our scenes and to get better results, we had to make some improvements. Using light maps and the ZP+ algorithm, we achieved very convincing results combining the strengths of the different approaches: the speed of the shadow mapping and the versatility of the shadow volumes technique. Due to the use of several methods, we got a wide range of rendering speeds. In the final stage, we modified the algorithm to tune the frame rate more finely. Thus we got a final pipeline that efficiently renders shadowed pictures at the required frame rate, regardless of the actual hardware possibilities. On fast computers, the final process produces high-quality pictures with several coloured light sources in real time. On slower configurations, our algorithm can either render with a better shadowing mode at interactive frame-rate or keep providing a real-time environment but with a lesser quality. This choice has to be done by the command application and can be changed on-the-fly.

This report is far from being exhaustive on the study of shadowing algorithms. First we do not take advantage of the capabilities of state-of-the-art hardware (e.g. pixel shading). We also focused on static scenes to implement an algorithm that meets our main expectations: render in real time on average configurations. However, there are still a lot of possible relevant improvements to get better shadowing possibilities. From an aesthetic point of view, some issues are not well solved and the proposed algorithms have to be modified. Moreover, the current rendering parameter determination is still very rough. Indeed, it is a classical “costs-benefits” optimization. Thus, the implementation of more effective algorithms to solve the pre-computation step and achieve a greater adaptability could be fruitful.

Bibliography

[1] Frank Crow. Shadow algorithms for computer graphics. *Computer Graphics : Proc. of SIGGRAPH 1977*, pages 242–248, 1977.

[2] Harlen Costa Batagelo and Ilaim Costa Junior. Realtime shadow generation using BSP trees and stencil buffers. *SIBGRAPI*, volume 12, pages 93–102, 1999.

[3] Michael McCool, Shadow Volume Reconstruction from Depth Maps. *ACM Transactions on Graphics*, pages 1-25, 2001.

[4] Stefan Brabec and Hans-Peter Seidel. Shadow volumes on programmable graphics hardware. *Computer Graphics Forum : Proc. of Eurographics 2003*, 2003.

[5] Cass Everitt and Mark J. Kilgard. Practical and robust stenciled shadow volumes for hardware-accelerated rendering. http://developer.nvidia.com/docs/IO/2585/ATT/RobustShadowVolumes.pdf, 2002.

[6] Maneesh Agrawala, Ravi Ramamoorthi, Alan Heirich, and Laurent Moll. Efficient image-based methods for rendering soft shadows. *Computer Graphics:Proc. of SIGGRAPH 2000,* pages 375–384. 2000.

[7] W. Heidrich, S. Brabec, and H.-P. Seidel. Soft Shadows Maps for Linear Lights. In *Rendering Techniques: Proc. of the 11th Eurographics Workshop on Rendering*, pages 269–280, 2000.

[8] Zhengming Ying, Min Tang, and Jinxiang Dong. Soft shadow maps for area light by area approximation. *Proc. of the 10th Pacific Conference on Computer Graphics and Applications*, pages 442–443. 2002.

[9] Steven Parker, Peter Shirley, and Brian Smits. Single sample soft shadows. *Technical Report UUCS-98-019, Computer Science Department, University of Utah*, 1998.

[10] Brabec, S., and Seidel, H. P. Single sample soft shadows using depth maps. *Graphics Interface: Proc. of GI 2002*, pages 219-228, 2002.

[11] Florian Kirsch and Juergen Doellner. Real-time soft shadows using a single light sample. *Journal of the Winter School on Computer Graphics 2003*, 2003.

[12] Cyril Soler and François X. Sillion. Fast calculation of soft shadow textures using convolution. *Computer Graphics :Proc. of SIGGRAPH 1998*, pages 321–332. 1998.

[13] Sylvain Vignaud. Real time soft shadows on geforce class hardware. http://tfpsly.planet-d.net/english/3d/SoftShadows.html, 2003.

[14] Eric Haines. Soft planar shadows using plateaus. *Journal of Graphics Tools*, pages 19–27, 2001.

[15] Eric Chan and Fredo Durand. Rendering fake soft shadows with smoothies. *Rendering Techniques 2003 : Proc. of the 14th Eurographics Symposium on Rendering*, 2003.

[16] Ulf Assarsson and Tomas Akenine-Möller. A geometry-based soft shadow volume algorithm using graphics hardware. *ACM Transactions on Graphics : Proc. of SIGGRAPH 2003*, 2003.

[17] Tomas Akenine-Möller and Ulf Assarsson. Approximate soft shadows on arbitrary surfaces using penumbra wedges. *Proc. of the 13th Eurographics Workshop on Rendering*, pages 309–318, 2002.

[18] Samuel Hornus, Jared Hoberock, Sylvain Lefebvre and John Hart. ZP+ : Correct Z-pass Stencil Shadows.

[19] J.M. Hasenfratz, M. Lapierre, N. Holzschuch and F.X. Sillion. A Survey of Real-time Soft Shadows Algorithms. *Artis GRAVIR/IMAG-INRIA*. 2003.

[20] Marc Stamminger and George Drettakis. Perspective Shadow Maps. *ACM Transactions on Graphics: Proc. of SIGGRAPH 2002,* 2002.

[21] Cass Everitt & Mark J. Kilgard. Optimized Stencil Shadow Volumes. *NVIDIA Corporation*, 2002.

[22] Greg James and John O'Rorke. Real-time glow. *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphic*, 2004.

[23] Céline Loscos and George Drettakis. Interactive High-Quality Soft Shadows in Scenes with Moving Objects. *EUROGRAPHICS '97*, 1997.

List of figures

List of algorithms

List of graphs

List of tables