# Offsety i operatory Minkowskiego w przyspieszaniu obliczeń globalnego modelu oświetlenia.

Andrzej Łukaszewski

praca doktorska

promotor: Prof. dr hab. Leszek Pacholski

Instytut Informatyki
Uniwersytet Wrocławski

Wrocław, 2001

# Offsets and Minkowski operators for speeding up global illumination methods.

**Andrzej Łukaszewski**

**Ph.D. Thesis**

**supervisor: Prof. dr hab. Leszek Pacholski**

**Institute of Computer Science**
**University of Wrocław**

**Wrocław, 2001**

# Acknowledgements

First of all, I would like to thank Prof. Leszek Pacholski for his support in doing research on computer graphics. I am obliged to him for helpping me in writing the thesis and for his help in establishing international contacts that made this research possible and influenced my work.

I would like to thank Prof. Hans–Peter Seidel from Max Planck Institüt für Informatik in Saarbrücken for collaboration and the time of work in his group when I put together the most part of this thesis. I also want to express my thanks to all members of his research group for many valuable discussions and the nice atmosphere during my stay there.

Wrocław, October 2001.

This work has been partly supported by

– Uniwersity of Wroclaw research grant nr 2203/W/IN/97i98 (I.1997–XII.1998)

– KBN research grant nr 7 T11C 023 20 (II.2001–XII.2001)

– Marie Curie Fellowship for stay in MPI in Saarbrücken (III.2001–VIII.2001)

# Contents

# Chapter 1

# Introduction

*"Somehow it seems to fill my head with ideas —*
*only I don't know exactly what they are !"*

Lewis Carroll — Through The Looking Glass

Computer graphics began with a need to visualize huge amounts of data and with the need to provide convenient interface between the human and the computer. In the seventies the hidden surface removal algorithms were developed together with the first local shading models [45]. These algorithms improved the three dimensional impression of generated images. Instead of wireframe images it was possible to visualize the scenes with correct visibility and simple simulation of light propagation.

Through the years the quality of images was getting better. One of the goals of computer graphics became synthesis of images as close to reality as possible. The first step to calculate the real global illumination was the ray tracing algorithm proposed in 1981 by T. Whitted [54]. In some setups of scenes the effects were so satisfying that the term photo–realistic image synthesis was created. It meant that synthesized images would be indistinguishable from real photographs. However, it was computationally quite expensive, therefore a lot of research was done through the next decades to accelerate the calculations. Better algorithms were developed and computational power of computers has been increased through the years. Currently, using ray tracing algorithm we are able to synthesize images of huge scenes consisting of millions of triangles relatively fast.

There has been also a lot of research in the area of algorithms to calculate physically correct simulation of light propagation. To get physically correct solution we have to solve the global illumination equation which describes transfer of light energy at each point. Two groups of methods have been developed to solve the global illumination equation: Monte Carlo methods (for an overview see Section 1.3) and finite element methods. Monte Carlo methods are stochastic methods that are based on the ray tracing algorithm. They sample the space of light paths to get an approximate solution. They share the same ideas of tracing rays with Whitted method mentioned above. On the other hand the finite element methods also use rays quite often to determine mutual visibility what is necessary for the calculation of so called form factors between patches. As we see ray tracing is important whenever we want to calculate illumination and to synthesize realistic images.

Monte Carlo ray tracing methods are the most popular and are used both in public domain graphics packages and in commercial products used by the movie industry. A single frame of a cinema movie constists of about $10^7$ pixels and is rendered from a scene which also consists of millions of objects. Therefore the cost of production of a movie is very high. Movie companies often use so called rendering farms. They consists of thousands of computers which are just used for rendering. Therefore even a small reduction of the cost of calculations can give substantial savings.

In computer graphics we use the term interactive image synthesis when several frames per second can be generated. The real time image synthesis is defined by human perception capabilities. For film and television standards the real time usually denotes 25–60 generated frames per second.

Monte Carlo methods are quite flexible and it is usually possible to get better images just by tracing more rays. They are also easily scalable. Let us consider an application where interactive or real time image generation is required. Any acceleration will allow us to generate more frames per second or by tracing more rays we can visualize the effects which would otherwise be visually missing. With every speedup of the basic method we will get more exact illumination solution.

In ray tracing literature there has been many ideas how to exploit coherence of rays and objects in the scene to reduce the cost of calculations. We present the summary of results in Section 1.4. Most of the methods for ray tracing acceleration were developed in the eighties. They usually exploit coherence in object space and construct geometric structures which allow fast determination of the first object hit by a ray. Usage of the coherence is essential for the method and in the common case of complex scenes is necessary. It would be far too expensive to find intersection with every object. The advances in this field together with growing computational power of computers made it possible to get to the point where the image synthesis can be done almost in the real time. However, synthesis of photo–realistic images close to reality in real time is still a challenge. To handle it, simplifying assumptions still have to be made or limits on the scene complexity have to be imposed. Calculating the complete illumination for a non trivial scene requires lots of computations if we include all the effects e. g., multiple reflections (both specular and diffuse), and soft shadows caused by non-point light sources. We are getting closer to the point when the global illumination can be calculated at interactive rates.

Shadow ray tests if the ray from the light to the given point is not obstructed by an obstacle. We are interested in generalized shadow rays. These are any rays between two points (none has to belong to the light source) that test if the light along the ray is not obstructed by an obstacle. They are widely used in visibility checking, and it is the most time consuming part of many algorithms for image synthesis. Monte Carlo methods which calculate global illumination are based on the ray tracing principles, and use shadow rays extensively. Shadow rays are also applied in some of the finite element methods to calculate form factors where visibility has to be determined.

We propose a novel technique which exploits the ray coherence in a new way. It works for generalized shadow rays. It is independent of classical acceleration methods based on object coherence. Therefore it can be used together with them and can give a significant reduction of costs.

Our method allows in some cases to answer visibility query for a group of rays in the cost of tracing one shadow ray. Therefore in some cases it eliminates the need of separate checks for different rays. Instead of tracing a bundle of rays we test just one ray in a modified scene and if the test succeeds we know that all the rays from this bundle are not obstructed. Our method is based on offsetting operation and its generalization using Minkowski operators.

This technique does not exclude using some recent methods based on studies of visual perception which guide the calculations into important regions and avoid calculating effects which are not perceived by humans.

## 1.1 Contributions of this thesis

In this thesis we present a new technique for speeding up ray tests. The general scheme is presented together with formal proof of correctness in Chapter 3. In Chapter 4 we give results of experiments in case of stochastically sampled area light sources.

Chapter 5 gives a new algorithm for intersecting an offset of a rational surface with a ray. It is substantially faster than the one previously known. This algorithm is usefull not only in the context of the technique presented in Chapter 3. It also makes possible to directly visualize offsets using ray tracing method and can be used in collison detection.

Parts of this work have already been published in [37, 36].

## 1.2 Organization of this thesis

This thesis is organized as follows. Chapter 1 gives an introduction and provides an overview of existing ray tracing and global illumination Monte Carlo methods. Special attention is given to the acceleration techniques and shadow generation. Chapter 2 introduces terminology. We also give definitions of offsetting and Minkowski operators. Chapter 3 presents lemmas which establish foundations of our new methods for acceleration of ray tracing algorithms. These results use offsetting and Minkowski operators. We state the lemmas which give the theoretical background and guarantee the correctness of our method. Chapter 4 describes the method for soft shadows and gives results of experiments.

Chapter 5 presents a technique of finding intersections of a ray with an offset of a rational surface. It is useful both for direct offset surface visualization and for acceleration technique given in Chapter 4.

## 1.3 Ray tracers and Monte Carlo methods

Ray tracing is a powerful rendering technique which simulates light propagation. The basic algorithm is simple and can be easily extended. It is more general than other methods for global illumination. The algorithm can handle different types of objects and enables implementation of different light effects.

Ray tracing is often considered as an expensive method, but in fact it is not true. It can be used for hidden surface removal. When used in complex environments which contain many objects the ray tracing outperforms $z$-buffer algorithm which is classicaly used for hidden surface removal. It is also quite promising for interactive and real time applications, where a time limit for rendering an image is given, and computational resources are limited. Due to scalability of ray tracing we can cast as many rays as we want and we calculate the best approximate illumination solution we can get. Therefore depending on computational power and time available we can get anything between an exact global illumination of a scene and a simple hidden surface removal with a simple constant shading.

### 1.3.1 Whitted ray tracer

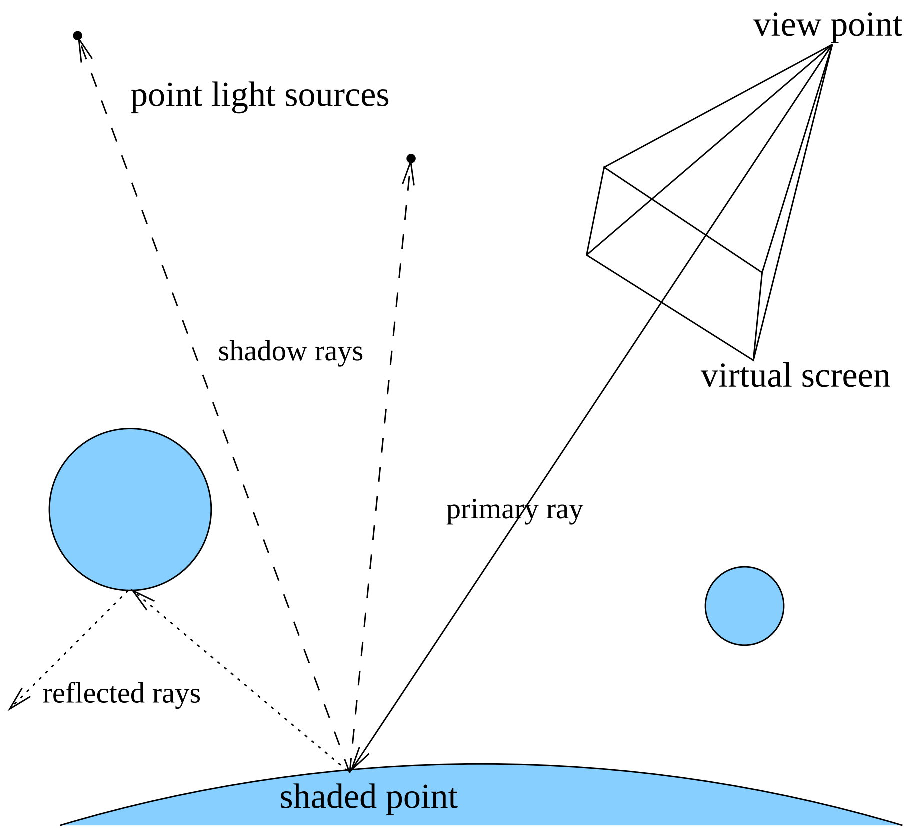

Figure 1.1: Ray tracing principle

The classical Whitted ray tracer [54] computes color of each pixel in the image by tracing a primary ray from the eye into the scene to find the nearest object visible (this method is called ray casting). On the first intersection it calculates outgoing light along the eye ray using given reflectance model. It adds components of direct incoming light from light sources and the light incoming along the direction of perfect specular reflection. In case of transparent materials we add the component for the light incoming from direction determined by diffraction. The last two factors are computed recursively by the same method of ray tracing to determine the light incoming along the ray. For direct components so called shadow rays are traced to the light sources to determine whether the point is not obstructed by any obstacle. The situation is illustrated on Figure 1.1. As it has been already noticed by Whitted most of the calculation time is spend on tracing the rays i. e., finding the nearest intersection of the ray with the scene. History and an overview of the ray tracing algorithms can be found in [14].

Classical ray tracing method has some severe drawbacks. The illumination model is simplified by the restricted choice of light paths and it does not calculate multiple reflections correctly except for perfect specular reflections. The method is also restricted to the point light sources without spatial extend which causes sharp shadows. It is not well suited for scenes with diffuse objects and diffuse light sources. As an example so called effect of color bleeding which is based on light transfer between two diffuse surfaces can not be simulated by Whitted ray tracer.

To calculate global illumination exactly it would be necessary for each point to collect and integrate information about incoming light from all directions not only from the few directons used in the Whitted algorithm. Monte Carlo methods estimate illumination function tracing finite number of rays selected in some random way depending on the method. The history and the review of Monte Carlo methods together with further references can be found in [51]. We will sketch here the most common examples.

### 1.3.2 Stochastic ray tracing

Stochastic[1] ray tracing introduced by R. Cook in 1984 [5] is a method where instead of one ray per reflection, refraction or light source, a bundle of random rays is generated. It is illustrated on Figure 1.2. At each intersection point this method generates many rays for which it is called recursively. At the cost of tracing a large number of rays it calculates effects which would be otherwise missing in a classical solution.

To implement spatial light sources and soft shadows we stochastically sample the light sources. For each point to be shaded, a certain number of rays is “fired” towards each of the light sources. The target points on the surface of a light source are

[1]formerly called distributed ray tracing, but this notion can be misleading suggesting parallel processing in distributed environments

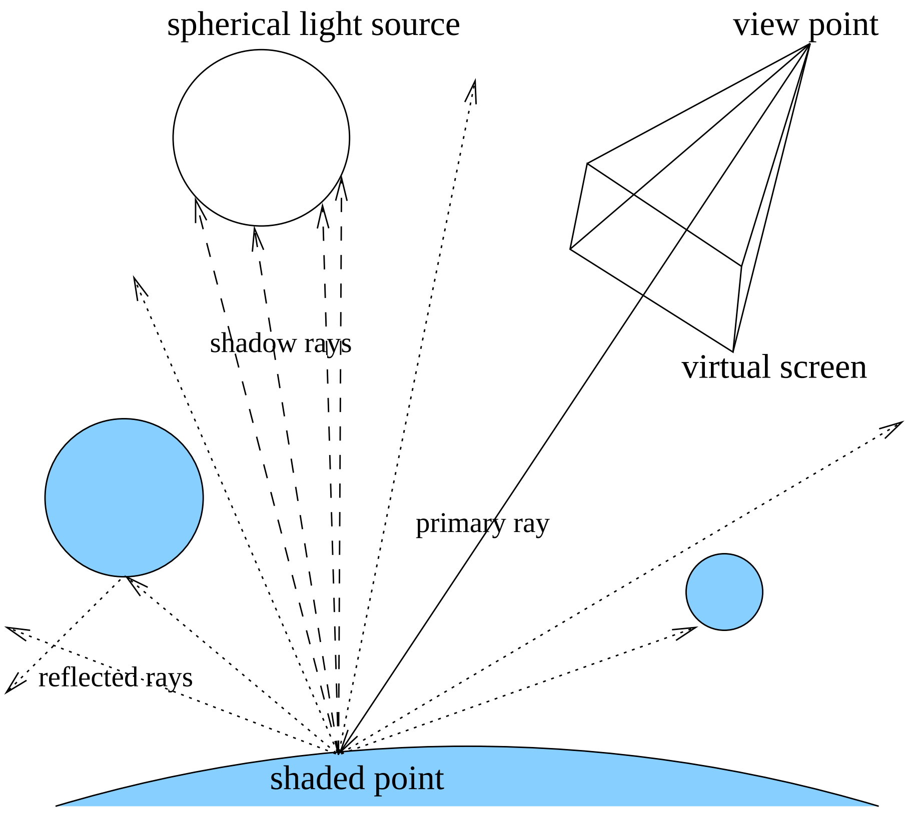

Figure 1.2: Stochastic ray tracing

distributed randomly implementing some kind of Monte Carlo integration method to estimate the correct size of the visible solid angle.

To get correct multiple reflections we replace one specularly reflected ray with multiple reflected rays which are usually generated using importance sampling according to the chosen local reflectance model described by the BRDF function (bidirectional reflectance distribution function).

### 1.3.3 Photon tracing and photon maps

The classical ray tracing as well as the stochastic method described above trace rays in the direction reverse to the direction of light propagation. It is simpler to realize since the rays are traced backwards taking into account what is seen by the observer.

Photon tracing method [26] uses the idea to trace light propagation in its physically correct direction. Some kind of importance sampling has to be used to guide the rays to the visible regions of interest. Photon tracing method was studied by Pattanaik in [41, 42, 43]. Photons are shot from the light sources and are traced into the scene. This is the phase of shooting. Then photons are stored in the scene where they are reflected or absorbed. Based on this information the global illumination solution is calculated by density estimation methods [49]. To estimate illumination in the given point we collect information about photons and energy from a neighbourhood of this point. If we collect more photons from a larger region we get a smooth solution but it is biased e. g., sharp shadows can become fuzzy. On the other hand taking a small neighbourhood results in bigger level of noise due to stochastic nature of the algorithm.

One of the methods is the photon maps developed by Jensen [25]. Photons are stored in the spatial kd-tree structure. This data structure allows to efficiently collect neighboring photons.

### 1.3.4 Bidirectional path tracing

The method explored by E. Lafortune et al. [32, 33] traces rays from both directions as it is shown on Figure 1.3. It constructs paths of photons from light sources and at the same time traces rays from the observer. It connects these paths and calculates light transports what gives global illumination solution. This method is expensive but it is more flexible than others. The other methods which trace the paths either only from

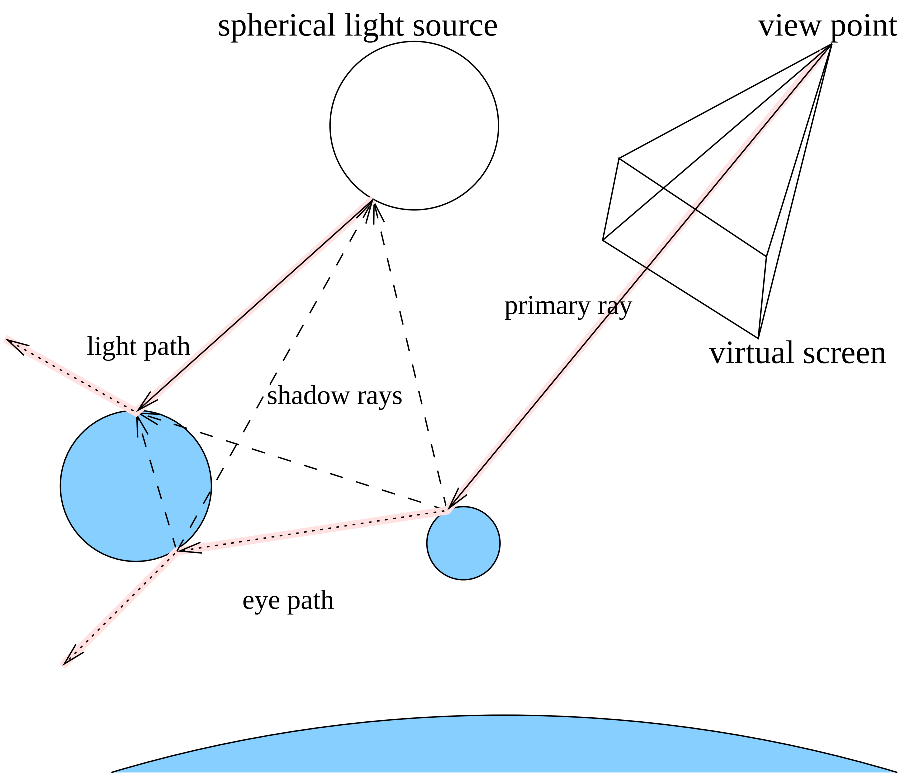

Figure 1.3: Bidirectional path tracing

the observer or only from light sources are special cases of bidirectional path tracing. Vertices of these two paths are connected by shadow rays. Each shadow ray is traced checking mutual visiblity of its end points. A very big number of shadow rays is one of the factors responsible for the cost of this method. In fact if we have the light path with $n$ vertices and the eye path of with $m$ vertices then we have to check the $\frac{n \cdot m}{2}$ shadow rays.

## 1.4 Acceleration techniques

Any method based on ray tracing principle has to be efficient in shooting rays and finding their intersections. Therefore the algorithms finding intersections for different types of objects should be efficient and have to be well tuned. Efficiency was always a critical issue for ray tracing. Therefore many acceleraton techniques have been developed. They are presented in the book [14] with detailed references. Here we will shortly recall these techniques.

Typical scenes consists of small objects and it is very probable that for a small object a given ray does not hit it. Therefore one of the first ideas was to enclose the groups of objects and more complex objects in so called bounding volumes. These are objects like spheres or boxes for which fast ray intersection algorithms exist. Now, the ray can be tested against such a bounding volume, and only if the ray hits the bounding volume the ray intersection with the objects inside the bounding volume has to be calculated.

The obvious extension is to enclose several bounding volumes in a bigger one creating a hierarchy of bounding volumes. This can give a significant reduction of the number of intersection tests and it is necessary for scenes with a huge number of objects. Such hierarchies are also called acceleration structures and are constructed on the base of data structures like octrees [15], BSP-trees [27] or kd-trees. They have to be optimized so that finding the objects which the ray can potentially intersect is fast. Therefore in construction of acceleration stuctures different heuristics are used. The comparison of different methods can be found in [18].

Uniform and nonuniform grids are also used to accelerate ray tracing (e. g., [13]). They divide space into rectangular regions in which the information about all objects intersecting the region is kept. The ray traverses the grid and for each grid element the objects kept there are checked whether they intersect the ray. In this method only objects in visited grid elements are checked.

The methods mentioned above use object space coherence. Using coherence of rays is less common. The few examples of this technique are the light buffer method developed by Haines and Greenberg [17] for shadow rays and the ray coherence method of Ohta and Megawa [39]. These methods reduce the cost but do not completly remove the need to trace each ray. Another approach in this context is the use of generalized rays, like cone tracing [2] or beam tracing [20]. However, since more complex geometric entities like cones or general pyramids are used, these methods often apply to quite restricted set of primitive objects and they require special intersection algorithms. Therefore they are not widely used.

Currently the best algorithms together with carefully designed implementations make it possible to render the images interactively [52].

## 1.5 Shadows

Realistic shadow generation plays an important role when producing computer generated images. The human observer is accustomed to see shadows in an illuminated scene in the real world, so shadows should be present in a computer generated image. If there are no shadows, or only sharp shadows where they are unappriopriate, the image is perceived as artificial. Moreover, shadows enhance the perception of the third dimension in the two–dimensional image [53].

The computation of shadows is a very expensive task for every rendering algorithm. Many simple rendering programs model light sources as mathematical points without any three-dimensional extension. Such light sources cause sharp shadows, because the shadow calculation reflects a step function: a point to be displayed is either in shade or in light (as long as we do not consider indirect illumination through diffuse or specular reflection on surfaces).

In real environments, however, the transition from illuminated to non-illuminated regions is smooth. A point is in shade respective to a certain light source when an obstacle totally occludes the light source. In other words, if any ray starting at the point and going towards the light source intersects an opaque surface before it reaches the surface of the light source. Conversely, a point is in light respective to a certain light source when the light source is entirely visible from the point. Penumbra occurs when an obstacle partially occludes the light source, allowing only a subset of the rays to reach the light source. Adding penumbras or so-called soft shadows makes the problem of shadow generation more complex. A survey of different techniques is given e. g. in [56].

Stochastic ray tracing is very easy to implement and delivers images of very high quality. However, it is computationally expensive. For each intersection point found in the scene, a large number of rays is sent towards all light sources. If we sample the visible solid angle of a light source with $d$ rays per point without any enhancement, the run time to trace the shadow rays is roughly $d$ times larger than the run time in ordinary ray tracing that generates sharp shadows. To achieve good image quality, the value of $d$ should depend on the size of the light source; values of $d \geq 50$ are often necessary. Several approaches are available to speed up stochastic ray tracing. The most important are the shadow buffer and importance sampling.

The shadow buffer, as introduced in [17] and extended in [44], is very important in stochastic ray tracing for penumbra. Because we send several rays from the same point to the same light source an object cached in the buffer is likely to serve as occluding object for many rays. However, the shadow buffer does not exhibit such a large improvement as one might expect at the first sight. Many rays for penumbra calculation pass close to the objects but they do not hit the objects. The shadow buffer is best to speed up tracing rays that actually hit objects. Thus, roughly speaking, only half of the rays sent out in a penumbra can profit from the shadow buffer. For points outside of any shadow region, there is no advantage of the shadow buffer.

Another method which uses importance sampling to reduce the number of rays sent out per point is presented in [31]. In this method rays are only traced in "important directions" that provide the main information necessary for adequate shading. However, one has to take care that the samples are generated properly to estimate the appropriate solid angle as seen from the point and to avoid adding too much noise to the image. The number of rays sent out can depend on the distance of the light source, their contributions to the illumination of the point, and some other parameters which depend on the geometry. For instance, the angle between the normal vector of a planar light source and the direction of the shadow rays can have an impact on the number of sample rays sent towards that light source.

Two other ideas to speed up tracing rays towards linear or planar light sources are described in [47, 3], see [55] for overview. To decrease the amount of work to be done, the candidate list of objects possibly intersected by the shadow rays is confined to the objects actually intersecting the cone from the point toward the area light source. The candidate list is generated dynamically. The approach can be seen as a special form of cone tracing [2] with shadow caching. An object is discarded when it does not intersect the light cone and it is put into the cache if it does.

Shadow photon map [24] is yet another method to calculate shadows efficiently in the context of photon tracing. The generation of soft shadows in image based rendering has been studied by [1] and [28]. There has been also some works to enable real time generation of soft shadows using graphical hardware e. g., [21, 19]. Although they often use similar ideas these solutions are out of scope of this work.

The method similar to ours has been later proposed independently by S.Parker et al. [40]. It uses some approximation of the soft shadow instead of slower but exact calculation. Their test for penumbra region detection is less general and more costly to implement than ours since it requires the intersection algorithms for all the objects to return also the minimal distance from the ray when it does not hit the object which is not required in our solution.

# Chapter 2

# Preliminaries

*"It always happens, said the Gnat."*

Lewis Carroll — Through The Looking Glass

We denote the set of real numbers by $\mathbb{R}$. We will use the notation $R(p, q)$ for the set of points of the ray segment which starts at the point $p \in \mathbb{R}^3$ and ends at the point $q \in \mathbb{R}^3$. For a given ray segment $R(p, q)$ we can define it in parametric form

$$R(t) = p + t(q - p) \quad for \quad t \in [0, 1] \tag{2.1}$$

The distance between two points $p$ and $q$ we will write as $d(p, q)$. The closed ball centered at the point $C$ and with radius $d$ will be denoted by $B(C, d)$.

We will denote the derivative of a function $f(t)$ by $\partial_t f(t)$ and partial derivatives in the same way e. g., for function $g(u, v)$ we have partial derivatives $\partial_u g(u, v)$ and $\partial_v g(u, v)$.

## 2.1 Bézier curves, surfaces and volumes

For representing three–dimensional objects we can use primitives like spheres, cones or cylinders or we can use simple point, line and face representations like polygonal meshes. However, these representatons are not perfect for smooth surfaces, more complex than sphere or plane. Parametric surfaces are used there and they are widely applied in computer aided design.

Bézier representation is fundamental for piecewise polynomial and rational parametric curves and surfaces. It was developed in late sixties for use in automobile industry independently by Pierre Bézier at Renault and by Paul de Casteljau at Citroen. They also presented the main properties of these objects. There are also other representations of curves ansd surfaces e. g., B–splines, Beta–splines, NURBS, Coon patches. We shall limit our research to rational Bézier surfaces which are general enough and very flexibile. They are also the most numerically stable ones as it was recently proved. More details about parametric curves and surfaces can be found in books [7] and [22].

In this section we define Bézier curves and surfaces using the notion of Bernstein polynomials and we will recall their basic properties. We shall use the notion of parametric curves for the functions having one dimensional domain independently of the dimension of the function range. The functions with two dimensional parameter domain we will call parametric surfaces and the functions having three dimensional domain we will call parametric volumes. We will also consider the surfaces and volumes which have one–dimensonal range i. e., their values are not points in $\mathbb{R}^k$ but numbers on the line $\mathbb{R}^1$.

**Definition 2.1 (Bernstein Polynomials)** *For a given integer $n$ there are $n+1$ Bernstein polynomials $B_{i,n}(t)$, for $i = 0, 1, \ldots n$, defined by :*

$$B_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i} \tag{2.2}$$

**Property 2.2** *For Bernstein polynomials the following properties hold.*

$$B_{0,n}(t) + B_{1,n}(t) + \ldots + B_{n,n}(t) = 1 \tag{2.3}$$

$$B_{i,n}(t) \geq 0 \;\; \textit{for} \;\; t \in [0,1] \tag{2.4}$$

$$B_{i,n}(t) = tB_{i-1,n-1}(t) + (1-t)B_{i,n-1}(t) \tag{2.5}$$

$$\partial_t B_{i,n}(t) = n\,(B_{i-1,n-1}(t) + B_{i,n-1}(t)) \tag{2.6}$$

**Definition 2.3 (Bézier Curve)** *Let us consider a set of control points $\{P_i : i = 0, 1, \ldots n\}$ in $k$–dimensional space $\mathbb{R}^k$. We define* Bézier curve of degree $n$ *by :*

$$C(u) = \sum_{i=0}^{n} B_{i,n}(u) P_i, \qquad for \qquad u \in [0,1]. \tag{2.7}$$

**Definition 2.4 (Bézier Surface)** *Let us consider a matrix of control points* $\{P_{ij} : i = 0, 1, \dots m,\ j = 0, 1, \dots n\}$ *in* $k$*–dimensional space* $\mathbb{R}^k$*. We define* Bézier surface of degree $(m, n)$ *as follows:*

$$S(u, v) = \sum_{i=0}^{m} \sum_{j=0}^{n} B_{i,m}(u) B_{j,n}(v) P_{ij}, \qquad for \qquad u, v \in [0, 1]. \tag{2.8}$$

We extend the definitions of Bézier curves and surfaces to three parameters and we present here the new notion of Bézier volumes.

**Definition 2.5 (Bézier Volume)** *Let us consider a set of control points* $\{P_{ijh} : i = 0, 1, \dots m,\ j = 0, 1, \dots n,\ h = 0, 1, \dots p\}$ *in* $k$*–dimensional space* $\mathbb{R}^k$*. We define* Bézier volume of degree $(m, n, p)$ *as follows:*

$$V(u, v, t) = \sum_{i=0}^{m} \sum_{j=0}^{n} \sum_{h=0}^{p} B_{i,m}(u) B_{j,n}(v) B_{h,p}(t) P_{ijh}, \qquad for \qquad u, v, t \in [0, 1]. \tag{2.9}$$

Now, we will recall some fundamental properties of Bézier curves and surfaces which also extend to Bézier volumes. Let us start with the convex hull property which gives simple means to determine the location of the object. From Equations (2.3) and (2.4) for Bernstein polynomials we have immediately the following.

**Property 2.6 (Convex Hull Property)** *Bézier curve or surface is included in the convex hull of the set of its control points.*

To refine the geometry of the curve we want to have the subdivision algorithm. That is we want to split the parametric Bézier curve into two pieces also represented as the Bézier curves. The algorithm invented by de Casteljau is the most fundamental and surprisingly simple. Due to its geometric nature it is very intuitive and numerically stable. We can derive it from definition of Bézier curve and Properties 2.2. It constructs the control points of the new curves and is illustrated in Figure 2.2.

**Property 2.7 (De Casteljau Algorithm for Curves)** *Consider Bézier curve* $C(t)$ *based on points* $\{P_i : i = 0, 1, \dots, n\}$*. We define auxiliary points using midpoint calculations as follows*

$$P_{i,0} = P_i \tag{2.10}$$

$$P_{i,j} = \frac{1}{2}(P_{i,j-1} + P_{i+1,j-1})\ ,\ \textit{for}\ \ j > 0 \tag{2.11}$$

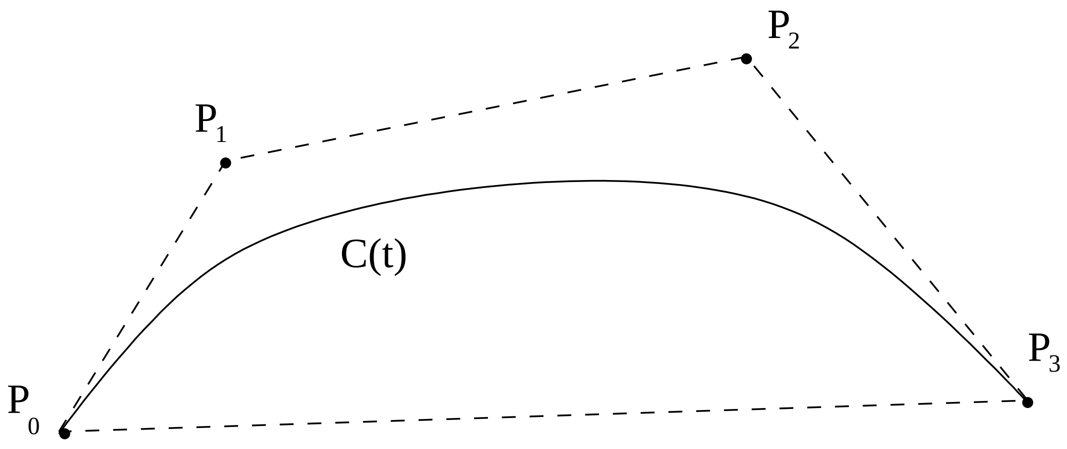

Figure 2.1: Convex hull property

*Than the Bézier curves $C_1(t)$ and $C_2(t)$ based respectively on control points $\{P_{0,i} : i = 0, 1, \ldots, n\}$ and $\{P_{i,n-i} : i = 0, 1, \ldots, n\}$ compose the original curve (for $t \leq 0.5$ we have $C(t) = C_1(2t)$ and for $t \geq 0.5$ we have $C(t) = C_1(2t-1)$) Therefore we have subdivided the original curve into two curves of the same degree. They correspond to parameter intervals $[0, 0.5]$ and $[0.5, 1]$ of the curve $C(t)$.*

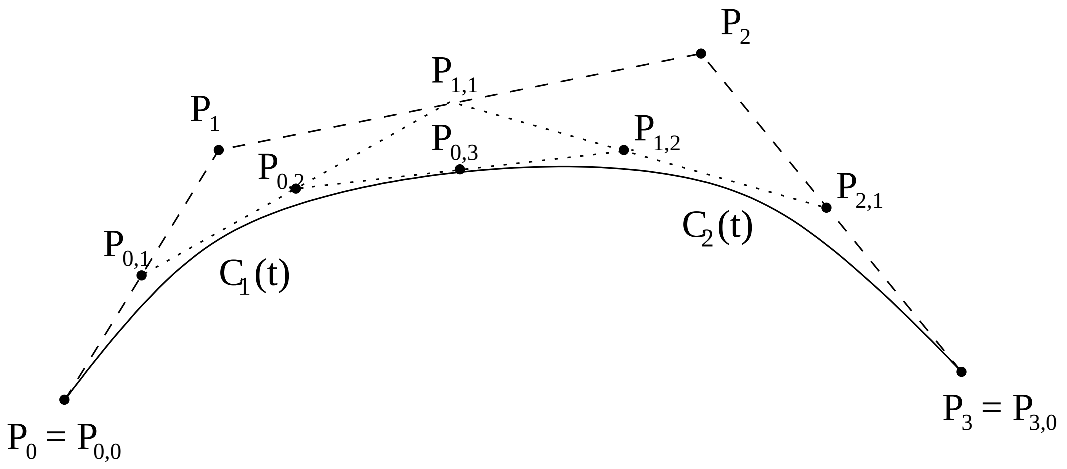

Figure 2.2: De Casteljau subdivision algoritm

The subdivision algorithm for a Bézier curve of degree $n$ requires $\frac{n(n+1)}{2}$ midpoint calculations. The cost of one midpoint calculation depends quadratically on the dimension of control points.

De Casteljau subdivision algorithm extends to surfaces and volumes. For a surface based on points $\{P_{ij} : i = 0, 1, \ldots m,\ j = 0, 1, \ldots n\}$ we split its parameter domain

$[0,1] \times [0,1]$ into four subdomains $[0,0.5] \times [0,0.5]$, $[0,0.5] \times [0.5,0]$, $[0.5,0] \times [0,0.5]$ and $[0.5,0] \times [0.5,0]$. Using de Casteljau algorithm for curves we obtain the control points of new Bézier surfaces corresponding to the above written domains of the original surface.

We use de Casteljau algorithm for curves to divide for, each $i = 0,1,\ldots,m$ the curve based on points $\{P_{ij} : j = 0,1,\ldots,n\}$ into two curves. This defines subdivision of the surface along one parameter direction. We obtained two sets of control points which define two new Bézier surfaces. Repeating this procedure for both these surfaces along the other direction will give us as a result subdivision of the original surface into four Bézier surfaces. The cost of the subdivision of the given degree $(m,n)$ surface is equal to the cost of $m+1$ subdivisions of degree $n$ curves and $2(n+1)$ subdivisions of degree $m$ curves what gives the total cost of:

$$(m+1)\frac{n(n+1)}{2} + 2(n+1)\frac{m(m+1)}{2} = \frac{1}{2}(m+1)(n+1)(n+2m)$$

If $m \neq n$ the cost depends on the choice of the first split direction.

Using similar procedure we can divide a Bézier volume along each parameter direction into two subvolumes defined by the control points obtained by de Casteljau algorithm from the original control points of the Bézier volume. Dividing in this way a volume along all three parameter directions we obtain eight subvolumes.

### 2.1.1 Bézier operations

We will show how to calculate control points of objects obtained by simple arithmetic operations on Bézier volumes.

**Fact 2.8 (Addition and difference of Bézier volumes)** *Let $F(u,v,t)$ and $G(u,v,t)$ be Bézier volumes of degree $(m,n,p)$ defined by points $F_{ijh}$ and $G_{ijh}$ respectively. Then $F(u,v,t) + G(u,v,t)$ and $F(u,v,t) - G(u,v,t)$ are Bézier volumes of the same degree $(m,n,p)$ defined by points $F_{ijh} + G_{ijh}$ and $F_{ijh} - G_{ijh}$ respectively.*

**Fact 2.9 (Multiplication of Bézier volume by a number)** *Let $F(u,v,t)$ be a Bézier volume of degree $(m,n,p)$ defined by points $F_{ijh}$ and let $d \in R$. Then $d \cdot F(u,v,t)$ is a Bézier volume of the same degree $(m,n,p)$ defined by points $d \cdot F_{ijh}$.*

**Proposition 2.10 (Multiplication of Bézier surfaces)** *Let $F(u,v)$ and $G(u,v)$ be Bézier surfaces of degree $(m,n)$ defined by points $F_{ij}$ and $G_{ij}$ respectively. Then $H(u,v) := F(u,v)\cdot G(u,v)$ is a Bézier volume of degree $(2m,2n)$ defined by points $H_{rs}$ :*

$$H_{rs} = \sum_{i+k=r}\sum_{j+l=s} \frac{\binom{m}{i}\binom{m}{k}}{\binom{2m}{i+k}} \frac{\binom{n}{j}\binom{n}{l}}{\binom{2n}{j+l}} F_{ij}G_{kl} \tag{2.12}$$

*Proof.* Using Definition 2.4 of Bézier surface we have

$$\begin{aligned}
F(u,v)G(u,v) &= \left(\sum_{i=0}^{m}\sum_{j=0}^{n} B_{i,m}(u)B_{j,n}(v)F_{ij}\right)\left(\sum_{i=0}^{m}\sum_{j=0}^{n} B_{i,m}(u)B_{j,n}(v)G_{ij}\right)\\
&= \sum_{i=0}^{m}\sum_{j=0}^{n}\sum_{i=0}^{m}\sum_{j=0}^{n}\binom{m}{i}\binom{m}{k}u^{i+k}(1-u)^{2m-(i+k)}\\
&\quad \binom{n}{j}\binom{n}{l}v^{j+l}(1-v)^{2n-(j+l)}F_{ij}G_{kl}\\
&= \sum_{i=0}^{m}\sum_{j=0}^{n}\sum_{i=0}^{m}\sum_{j=0}^{n}\frac{\binom{m}{i}\binom{m}{k}}{\binom{m+m}{i+k}}\frac{\binom{n}{j}\binom{n}{l}}{\binom{n+n}{j+l}}B_{i+k,2m}(u)B_{j+l,2n}(v)F_{ij}G_{kl}
\end{aligned}$$

□

We will be also interested in taking square of Bézier volume of degree $(m,n,1)$. We can represent such a Bézier volume $F(u,v,t)$ as the sum of Bézier surfaces of degree $(m,n)$ as follows.

$$F(u,v,t) = (1-t)\cdot S_0(u,v) + t\cdot S_1(u,v)$$

In that case to calculate $H(u,v,t) = F(u,v,t)^2$ we can write

$$\begin{aligned}
F(u,v,t)^2 &= ((1-t)S_0(u,v) + t(S_1(u,v))^2 =\\
&= (1-t)^2S_0(u,v)S_0(u,v) + (1-t)t\cdot S_0(u,v)S_1(u,v) + t^2S_1(u,v)S_1(u,v)
\end{aligned}$$

From the above equation we can see the following correspondence of control points: control points of the surface $S_0\cdot S_0$ are the points $H_{ij0}$, control points of the surface $2\cdot S_0\cdot S_1$ are the points $H_{ij1}$ and control points of the surface $S_1\cdot S_1$ are the points $H_{ij2}$. Therefore we have the following corollary.

**Corollary 2.11 (Square of Bézier volume of degree $(m,n,1)$)** *Let $F(u,v,t)$ be the Bézier volume of degree $(m,n,1)$ defined by points $F_{ijk}$. Then $H(u,v,t) = F(u,v,t)^2$*

*is a Bézier volume of degree* $(2m, 2n, 2)$ *defined by control points* $H_{rst}$ *as follows.*

$$\begin{aligned}
H_{rs0} &= \sum_{i+k=r}\sum_{j+l=s} \frac{\binom{m}{i}\binom{m}{k}}{\binom{2m}{i+k}} \frac{\binom{n}{j}\binom{n}{l}}{\binom{2n}{j+l}} F_{ij0}F_{kl0} \\
H_{rs1} &= 2 \cdot \sum_{i+k=r}\sum_{j+l=s} \frac{\binom{m}{i}\binom{m}{k}}{\binom{2m}{i+k}} \frac{\binom{n}{j}\binom{n}{l}}{\binom{2n}{j+l}} F_{ij0}F_{kl1} \\
H_{rs2} &= \sum_{i+k=r}\sum_{j+l=s} \frac{\binom{m}{i}\binom{m}{k}}{\binom{2m}{i+k}} \frac{\binom{n}{j}\binom{n}{l}}{\binom{2n}{j+l}} F_{ij1}F_{kl1}
\end{aligned}$$

From Equation 2.6 we can compute the derivative of a Bézier curve.

$$\partial_t \left( \sum_{i=0}^{n} B_{i,n}(t) P_i \right) = \sum_{i=0}^{n-1} B_{i,n-1}(t)\ n(P_{i+1} - P_i) \tag{2.13}$$

Thus we have the following fact.

**Fact 2.12 (Partial derivatives of Bézier volumes)** *If* $F(u, v)$ *is a Bézier volume of degree* $(m, n, p)$ *based on points* $P_{ijk}$ *then* $\partial_u F(u, v, t)$ *is a Bézier volume of degree* $(m-1, n, p)$ *based on the points* $R_{ijk} = m(P_{i+1,j,k} - P_{ijk})$ *and* $\partial_v F(u, v, t)$ *is a Bézier volume of degree* $(m, n-1, p)$ *based on the points* $T_{ijk} = n(P_{i,j+1,k} - P_{ijk})$*.*

### 2.1.2 Regular surfaces and normal vectors

The normal vector is a vector of unit length which is orthogonal to the surface in a given point. We shall give a formula defining it for regular surfaces. Therefore let us start with a definition.

**Definition 2.13 (Regular Surface)** *A parametric surface* $S(u, v)$ *is regular in its domain if for each point* $(u, v)$ *of the domain the partial derivatives* $\partial_u S(u, v)$ *and* $\partial_v S(u, v)$ *are not equal to zero and are not collinear.*

For a regular parametric surface $S(u, v)$ we can define normal vector as follows. The partial derivatives of the surface $S(u, v)$ in $u$ and $v$ directions are the vectors tangent to the surface. Therefore by the definition of the vector product (denoted here by $\times$) we have:

**Definition 2.14 (Normal Vector)** *For a regular parametric surface*

$$S(u, v) \; : \; [0, 1] \times [0, 1] \to \mathbb{R}^3$$

*the vector*

$$n(u, v) = \partial_u S(u, v) \times \partial_v S(u, v)$$

*is unnormalized orthogonal vector to the surface in the given point. We define the normal vector to the surface in a point* $(u, v)$ *as:*

$$N(u, v) = \frac{n(u, v)}{||n(u, v)||}$$

### 2.1.3 Rational Bézier surfaces

**Definition 2.15 (Rational Bézier Surface)** *We define* A rational Bézier surface of degree $(m, n)$ *is a parametric surface defined by:*

$$S(u, v) = \left( \frac{X(u, v)}{W(u, v)}, \frac{Y(u, v)}{W(u, v)}, \frac{Z(u, v)}{W(u, v)} \right) \tag{2.14}$$

where $X(u, v)$, $Y(u, v)$, $Z(u, v)$, $W(u, v)$ are Bézier surfaces of degree $(m, n)$ with the one–dimensional range.

To calculate the normal vector of a parametric surface it is sufficient to have well representated unnormalized vector orthogonal to the surface. It is constructed by means of partial derivatives and vector product. Following Boehm (see [4]) we represent it as a polynomial Bézier surface $n(u, v)$ of degree $(3m - 1, 3n - 1)$ with vector values

$$n(u, v) = (n_x(u, v), n_y(u, v), n_z(u, v))$$

where

$$\begin{aligned}
n_x &= (\partial_u Y \partial_v Z - \partial_v Y \partial_u Z)W + (\partial_v Y Z - Y \partial_v Z)\partial_u W + (Y \partial_u Z - \partial_u Y Z)\partial_v W \\
n_y &= (\partial_u Z \partial_v X - \partial_v Z \partial_u X)W + (\partial_v Z X - Z \partial_v X)\partial_u W + (Z \partial_u X - \partial_u Z X)\partial_v W \\
n_z &= (\partial_u X \partial_v Y - \partial_v X \partial_u Y)W + (\partial_v X Y - X \partial_v Y)\partial_u W + (X \partial_u Y - \partial_u X Y)\partial_v W
\end{aligned}$$

To shorten the formulas above we have skiped function parameters, and we have written $X$ for $X(u, v)$, $Y$ for $Y(u, v)$, and so on. Using equations above we can represent $n(u, v)$ as a Bézier surface by calculating its control points.

## 2.2 Minkowski operators and offsets

### 2.2.1 Minkowski operators

We shall study regions that consists of points that are close to given objects or included in these objects. For this purpose we shall define expansions and shrinking of objects using Minkowski operators. In our method we shall apply these operations for original objects casting shadows in a geometric scene.

Minkowski operators (e. g., [29]) provide a convenient way to express arithmetic operations on sets.

**Definition 2.16 (Minkowski Sum and Difference)** *For two subsets $A$ and $B$ of $\mathbb{R}^k$,* Minkowski sum *and* difference *are defined as:*

$$A \oplus B = \{a + b \,:\, a \in A,\ b \in B\} \ , \tag{2.15}$$

$$A \ominus B = \{a - b \,:\, a \in A,\ b \in B\} \ . \tag{2.16}$$

We shall use also scaling of a set by a scalar. We shall call it the Minkowski scaling.

**Definition 2.17 (Minkowski Scaling)** *For a subset $A$ of $\mathbb{R}^k$ and a real $c$,* Minkowski scaling *is defined as:*

$$c \cdot A = \{c \cdot a \,:\, a \in A\} \tag{2.17}$$

We shall use the notion of a star convex set, which is a weakening of the notion of a convex set.

**Definition 2.18 (Star Convex Set)** *We say that a subset $A$ of $\mathbb{R}^k$ is* star-convex *with respect to a point $c \in A$ if for any point $p \in A$ the segment $R(c, p)$ is included in $A$, i. e., if each point of $A$ can be connected to the point $c$, called a center of $A$, by a segment included in $A$.*

### 2.2.2 Solid offsets

Solid offsetting of sets is an expansion operation (see [8, 48]). It is a special case of using Minkowski operators.

**Definition 2.19 (Solid offset)** *Let $B(p,d)$ denote the ball with the center $p$ of radius $d$. The solid offset of the set $Q$ with the distance $d \geq 0$ is defined by:*

$$\mathcal{O}_d(Q) = Q \ominus B(0,d) = Q \oplus B(0,d) \quad . \tag{2.18}$$

We can use either $\ominus$ or $\oplus$ since the ball $B(0,d)$ is symmetric with respect to the point $0$. The idea is illustrated in Figure 2.3. The notion of a solid offset can also be defined as follows.

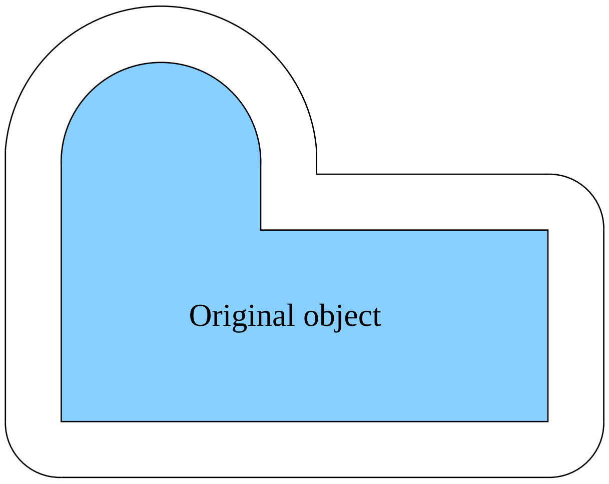

Figure 2.3: Solid offsetting operation

**Proposition 2.20 (Solid Offset)** *For an object $Q$ and a distance $d$, a* solid $d$-offset *$\mathcal{O}_d(Q)$ consists of the points that are not farther than $d$ from $Q$, i. e.,*

$$\mathcal{O}_d(Q) = \{p \ : \ \exists\, q \in Q : d(p,q) \leq d\} \quad . \tag{2.19}$$

The solid offsets define the operation of expanding objects. We can also use similar method to define shrinking of objects. Let $\overline{A}$ denote the set complement of $A$.

$$\overline{A} = \{x : x \notin A\}. \tag{2.20}$$

We define negative solid offset as follows.

**Definition 2.21 (Negative solid offset)** *Let $B(p,d)$ denote the ball with the center $p$ of radius $d$. The negative solid offset of the set $Q$ with the distance $d \geq 0$ is defined by:*

$$\mathcal{U}_d(Q) = \overline{\overline{Q} \ominus B(0,d)} \tag{2.21}$$

### 2.2.3 Offsets curves and surfaces

In the context of extending parametric surfaces by solid offsetting we shall define offset surfaces. They are surfaces on the the boundary of the solid offset of a parametric surface. The idea of offsets[1] was already introduced by Leibniz [30] for curves. Offsets are are important tools in robotics, computer aided geometrical design, geometrical optic and tolerance analysis (see [48]).

In machine milling the path of the cutter lies on the offset of the surface. Therefore calculation and interrogation of offsets is used in many places in numerical controled machine milling. Another application is configuration space approach used in robotics. To realize collision avoidance we use the model of the robot shrinked to the point, extending obstacles by offseting instead.

**Definition 2.22 (Offset Curve)** *For a planar curve $C(t)$ with the well defined normalized orthogonal vector $N(t)$ we define the offset curve at the distance $d$ as:*

$$C_d(t) = C(t) + d \cdot N(t) \tag{2.22}$$

*It is a displacement of the original curve in the direction of the normal vector.*

Figures 2.4 and 2.5 show examples of offset curves to parabolic and quadratic curves.

**Definition 2.23 (Offset Surface)** *For a regular progenitor surface $S(u, v)$ we define the offset surface at the distance $d$ as:*

$$S_d(u, v) = S(u, v) + d \cdot N(u, v) \tag{2.23}$$

*It is a displacement of the original surface in the direction of the normal vector.*

Chapter 15 of the book [22] gives the basic properties of offsets. The survey of main properties of offset curves can be also found in [10] and [11]. The results on approximation of offsets are presented in [6, 23, 9].

The offsets are much more complicated than the original curves or surfaces and usually do not belong to the same class as their progenitors (offsets of cubic curves are not cubic in general). Generally, even for simple polynomial curves and surfaces

[1]also called parallel curves and surfaces

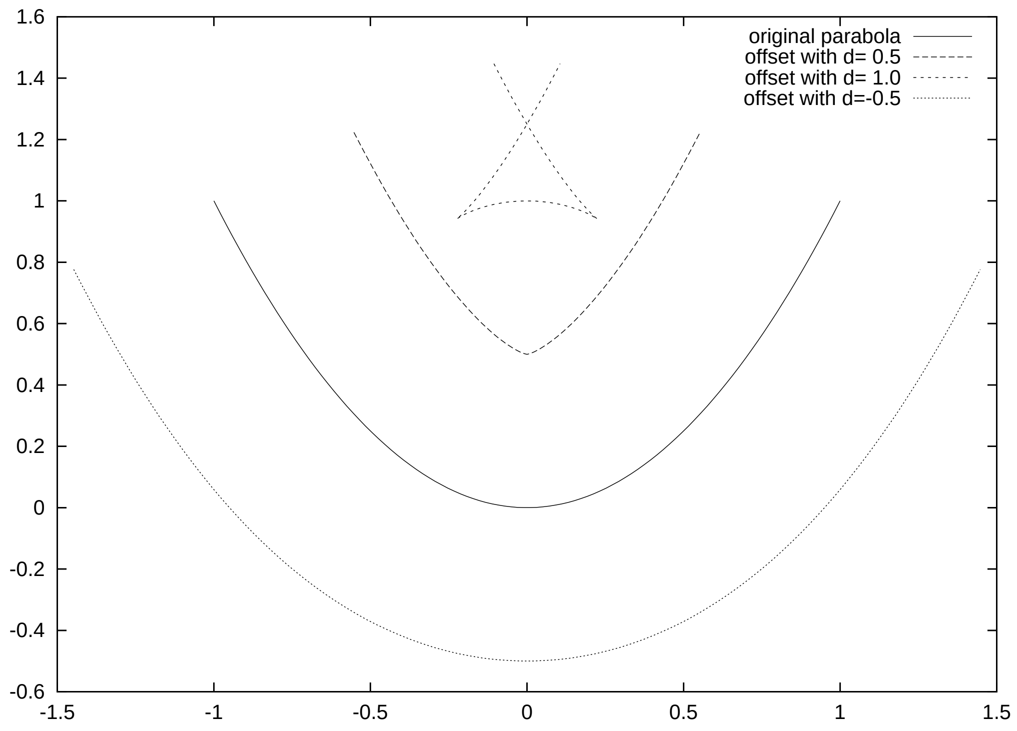

Figure 2.4: Offsets to parabola

their offsets are not rational and they are inherently more complex than the progenitor surfaces. As it can be seen on Figures 2.4 and 2.5 even for simple shapes offset curves have self–intersections. The same concerns offset surfaces. It happens when an offset distance is greater than the curvature radius of the curve or surface.

There are classes of rational surfaces with rational offsets. An example of such a subclass of rational Bézier surfaces which have rational Bézier offsets has been introduced in [46]. If we restrict ourself to such a class of surfaces the shapes of offsets are more simple and the cost of offseting operation becames smaller. However, in that case we can use only a small class of surfaces.

One of the basic algorithmic problems is finding intersection of an offset with a ray segment. It is used in collision detection in robotics, in machine milling, and for rendering an offset. An intersection algorithm of an offset of a polynomial Bézier surface with a ray was presented in [50]. It was also used to visualize offsets using the ray tracing method. The algorithm presented in chapter 5 (see also [37]) is substantially faster and works for rational surfaces as well.

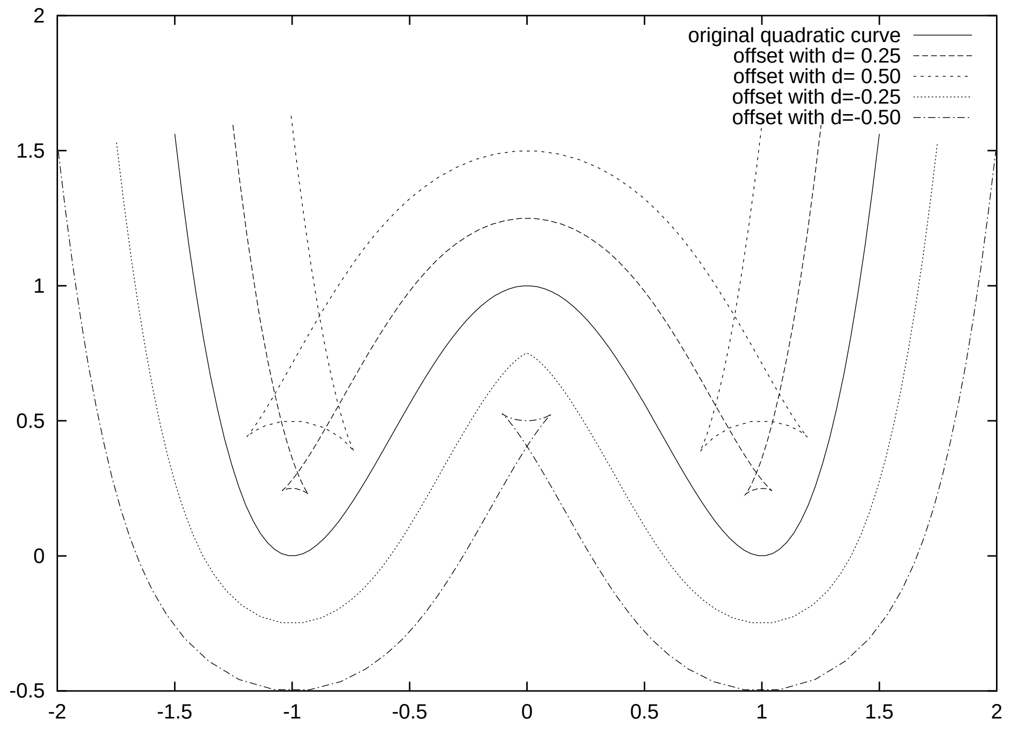

Figure 2.5: Offsets to quadratic curve

# Chapter 3

# Exploiting ray coherence

*"It's my own invention."*

Lewis Carroll — Through The Looking Glass

Spatial coherence of objects is commonly used in acceleration structures to make the ray tracing algorythm efficient. It is less common in ray tracing to exploit also spatial coherence of rays (see Section 1.4). Methods which use the coherence of rays limit the number of objects against which a ray has to be tested but they do not remove the need for separate ray tests. The cost of calculation of intersections is therefore reduced but each ray is tested for an intersection with a scene. Although the acceleration structures are used and the ray is not tested against many objects the total cost is significant.

The lemmas here present the general background how to check the visibility of a group of coherent ray segments in the cost of tracing a single ray segment. Therefore by tracing the single ray segment, in the cost reduced by any classical acceleration methods, we get the answer for the whole group of rays. Usage of the method is somehow limited to testing only visibility along the ray without calculation of first intersection. However, it can be used with most of other acceleration methods like grids, octrees, BSP-trees, kd–trees or hierarchical bounding boxes (see [14]) to further reduce their time requirements.

## 3.1 Rays with one common origin

Recall that there are three kinds of rays. Primary rays have origin in the eye of the observer. Reflected rays have origin at any point in the scene and either hit arbitrary point in the scene or go to infinity. Shadow rays originate in the point we are shading and have the other end points in light sources.

After reflection the rays loose coherence therefore we are not interested in reflected rays. The other cases worth examinination are as follows.

**Shadow rays** For non–point light sources we stochastically sample them to approximate the visibility angle. Therefore we have the family of rays with common point which we are shading. The other ends of rays belong to the light source so they are spatially coherent.

**Primary rays** We have the group of rays with the same origin and which can be generated coherently. However, the usage of the method here would be limited since in most of the cases rays intersect the scene. Therefore testing if the bundle of rays is not obstructed is less efficient. It will not eliminate the need for separate ray intersection tests and the calculation of hit points in most of the cases. Another weak point is that we do not know the ends of ray segments in advance, although we can construct them on the base of the bounding box of the scene. There are also specialized methods for speeding up the first hit intersection.

We propose the method in the case of rays with common origin. It is based on following lemma which is illustrated in Figure 3.1. When we apply it to speed up stochastic sampling of area light sources the set $L$ will denote the light.

**Lemma 3.1** *Let $L$ be a set star-convex with respect to a point $c \in L$. If a ray $R(c, p)$ does not intersect $Q \ominus (L \ominus \{c\})$ then for each $l \in L$ the ray $R(l, p)$ is not occluded (shadowed) by $Q$.*

*Proof.* (by contradiction) Let $C = \bigcup_{x \in L} R(x, p)$ denote the visibility cone of the set $L$ as seen from point $p$. If there would be a ray $R(l, p)$ obstructed by $Q$ then $C \cap Q \neq \emptyset$ but since the set $L$ is star-convex respective to the point $c$ we have $C \subset R(c, p) \oplus (L \ominus \{c\})$. Thus, we have $(R(c, p) \oplus (L \ominus \{c\})) \cap Q \neq \emptyset$ which means that there are points $r \in R(c, p)$, $l \in L$, and $q \in Q$ such that $r + (l - c) = q$. But this is equivalent to $r = q - (l - c)$ which means that $R(c, p) \cap (Q \ominus (L \ominus \{c\})) \neq \emptyset$. Hence, the ray does intersect the expanded object.

$\square$

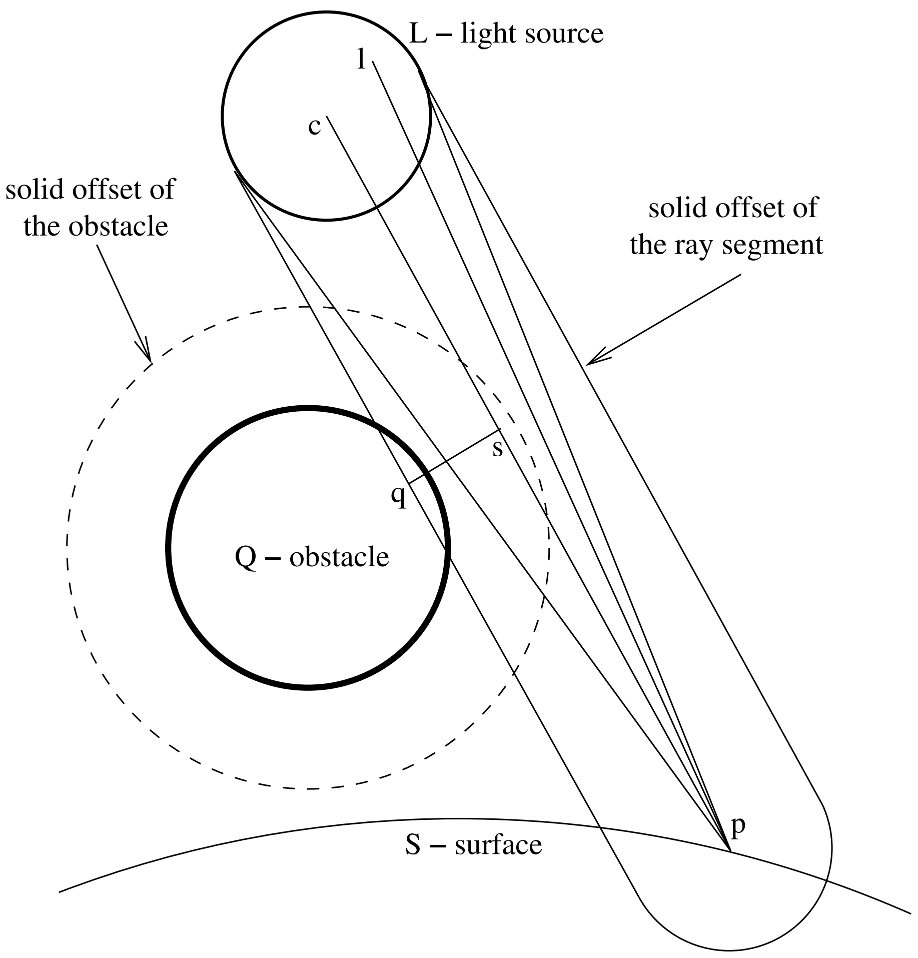

Figure 3.1: Not-in-shade condition for spheres.

Let us assume that the set $L$ is a ball of radius $d$. With the notion of solid offsets, we obtain that the bundle of ray segments starting in $p$ and having the other endpoint in $L$ is not obstructed by an object $Q$ if the ray $R(c, p)$ does not intersect the solid offset $\mathcal{O}_d(Q)$.

## 3.2 Rays with coherent origins

We can use the same framework as for rays with common origin for exploiting coherence of general shadow rays. That is, for those ray segments that we only test whether they are occluded or not. In bidirectional path tracing method we get large number of shadow rays between two paths. This method is feasible in that case. However, we have to cache the shadow rays and group them for testing. Let us assume that we have the set of rays with origins in the balls $A$ and $B$ of radius $d$. Then we can also guarantee that none of the rays is occluded by $Q$ if the ray $R(p, q)$ does not intersect $Q$ increased by $\mathcal{O}_d$ offsetting. The following lemma is illustrated by Figure 3.2.

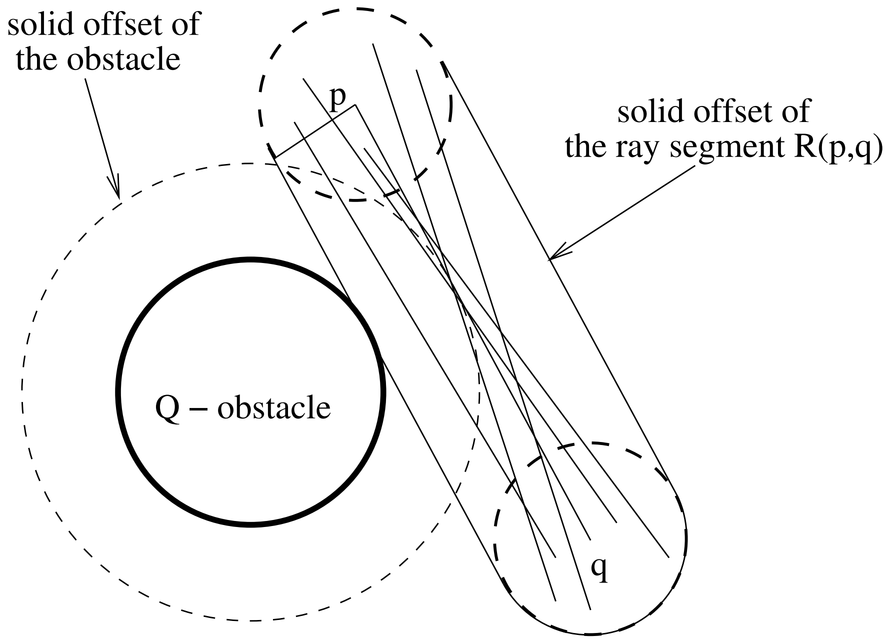

Figure 3.2: Bundle of coherent rays.

**Lemma 3.2** *Let $p, q \in \mathbb{R}^3$, and let $d \in \mathbb{R}$, $d > 0$. If the ray $R(p, q)$ does not intersect $Q \ominus B(0, d)$ then for each $r \in B(p, d)$, and each $s \in B(q, d)$ the rays $R(r, s)$ do not intersect $Q$.*

*Proof.* (by contradiction) Assume that ray $R(r, s)$ is obstructed by $Q$. Then since

$$R(r, s) \subset R(p, q) \oplus B(0, d)$$

we have

$$(R(p, q \oplus B(0, d)) \cap Q \neq \emptyset.$$

This is equivalent to

$$R(p, q) \cap (Q \ominus B(0, d)) \neq \emptyset,$$

which contradicts the assumption of the lemma.

□

Using this lemma we can trace at once the whole bundle of rays which have both the starting and ending points coherent and included in respective balls of a given radius. To check that none of the rays is obstructed by an object $Q$ from the scene we test just one ray in extended scene, only if the test fails then we have to do the normal tests for each ray, or try the test once again for a smaller bundle of rays. If the test succeeds we are often $n$ times faster, where $n$ is the number of rays in a bundle, since the cost in most of the cases is the same.

# Chapter 4

# Fast penumbra method

*"He did his very best to make*
*The billows smooth and bright —*
*And this was odd, because it was*
*The middle of the night."*

Lewis Carroll — Through The Looking Glass

We present a fast method to generate penumbras which avoids unnecessary calculations. It is based on stochastic ray tracing (see Section 1.3.2). There are no severe restrictions on the shape of the objects or the light sources. However, certain types of objects and light sources will allow faster rendering times. The main idea is to detect possible regions where penumbra occurs and to confine the expensive process of stochastic sampling of spatial light sources to those regions. Lemma 3.1 gives a condition, such that if it holds for a point we are shading, then we know that the light source is fully visible. Therefore we can skip tracing shadow rays and directly calculate illumination at this point.

## 4.1 Overview of the algorithm

We shall first describe our algorithm in a world of spheres. We model both the light sources and the objects as simple spheres. Figure 4.1 shows a scene where a light source $L$ casts a shadow on the surface $S$ because the light is occluded by the object $Q$. The umbra and the penumbra compose the entire shadow. The basic idea of the algorithm to speed up penumbra calculation is very simple. We detect penumbra regions and employ expensive calculations only when it is necessary. The detection is based on the following observation. If we shrink the light source $L$ to a point and, at the same time [1] , increase the occluding object $Q$ by the radius of $L$, then the true shadow volume is a subset of the approximate shadow volume. This depends neither on the radius of $L$ nor on the distance between $L$ and $Q$ or $L$ and $S$.

Lemma 3.1 gives the theoretical background for the general case. Description of the expansion is given by Minkowski operators. Lemma 3.1 also guarantees that the algorithm can work with objects and light sources of arbitrary shape and renders correct images.

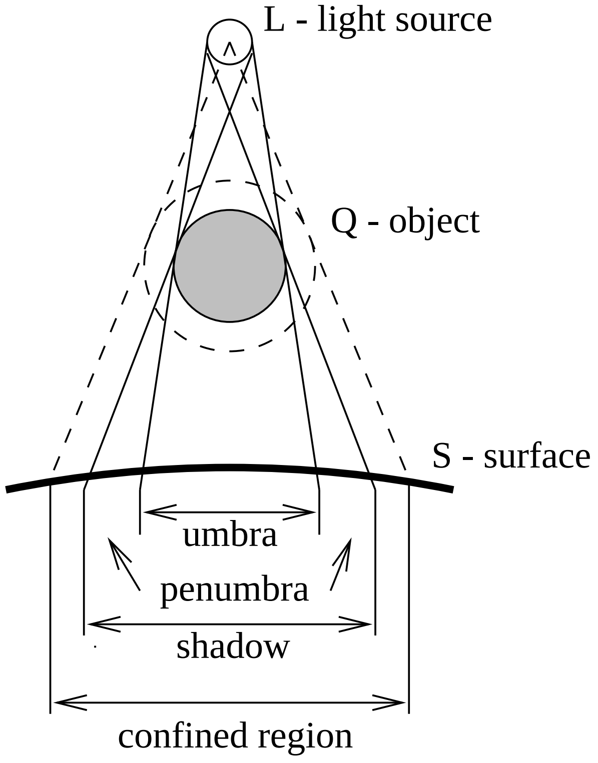

Figure 4.1: Shadow classification.

Once we have confined the shadow, we can employ analytical models or stochastic

[1]It is worth mentioning that the approach of using Minkowski operators on the objects of the geometric data set is also known in motion planning as a "configuration space approach" [29]. There, *C-obstacles* are defined that confine the areas which cannot be reached by a center point of a robot. Motion planning is performed in many stages of the algorithm only with this center point, so not the entire possibly complex robot must be considered in large areas of the environment.

ray tracing to sample the solid angle under which the light source is visible. Outside of the confined region we can skip this step since we know that the light source is fully visible. The following discussion is restricted to stochastic ray tracing which is used in our implementation.

For the ray tests which confine the region of penumbra we need the special scene with extended objects. Therefore we perform ray tracing in two different data sets. The "geometric" data set contains the environment as usual; the "shadow" data set contains the shrunken light sources and the increased objects. We determine in the shadow data set whether a given point belongs to a shadow region or not, i. e., shadow rays are initially traced in the shadow data set. If the point is in light, we apply the appropriate illumination model. If the point is found to be in shade, we start stochastic ray tracing in the geometric data set. As a further optimization, we detect umbra regions with a similar approach confining further the penumbra region (see Section 4.6).

Because there are two data sets, the memory requirements of the algorithm are at most twice as high as those of the the underlying ray tracing algorithm without the soft shadow enhancement. However, in the second set we only have to store the bounding volumes and the data structure used to accelerate ray tracing (grid, octree, or whatsoever). An intersection test in case of a bounding volume test in the shadow data set can be performed with an expanded object using its original geometrical description. The next section describes this concept in more detail. Discussion of the details of the algorithm follows.

## 4.2 Multiple light sources

Our method can handle multiple light sources. We create for each spatial light source the additional scene in which we perform the test for given shadow rays. We can also create just one expanded scene for all shadow ray tests taking into account the biggest expansion. If the light sources are of the similar size taking only one scene is as efficient as multiple ones and is less memory consuming.

As it is shown in Figure 4.2 we can use only the biggest expansion which is marked with dotted circles. In our implementation when more than one light source is present, we increase all objects by the maximum amount required by all the light sources. We use just one additional shadow scene.

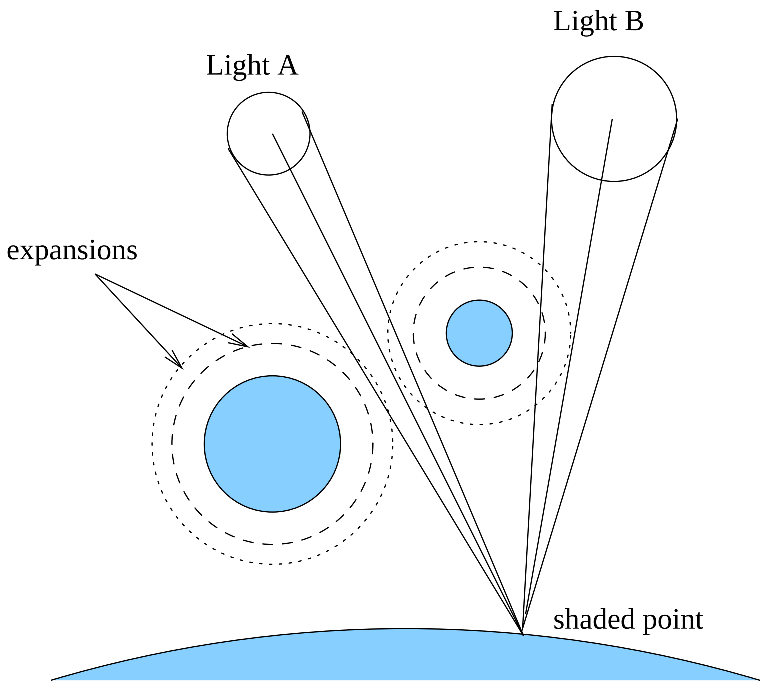

Figure 4.2: Multiple lights.

## 4.3 How to expand ?

Minkowski operators and offsets are hard or costly to evaluate in the general case. However, the not-in-shade condition is the base for many ideas to calculate bounding boxes or approximations of expanded objects.

Lemma 3.1 allows us to trace the collection of coherent rays at the cost of one intersection since the scene with increased objects can be constructed during the preprocessing stage. For spheres the offsetting operation is just increasing its radius but for other objects it can be expensive and can give more complex objects. However, we can always take a simpler object which includes our expanded one or to do increasing of just the bounding objects. Implementations can use couple of methods. In the second modified scene with expanded objects we can create either

- Only bounding boxes of expanded objects and acceleration structures for ray tracing. Then If the bounding box is hit we do calculations in original scene. This is quite universal and easy to implement.

- The exact expanded objects or larger ones with their bounding boxes and acceleration structures. If we construct exact expansion as given in Lemma 3.1 our test will succeed in more cases. We can also construct larger expanded object. If the test fails and there is an intersection we have to do ray tests in the original scene to guarantee the same results or to use other approximate methods.

For expanded objects the intersection test should not be significantly more expensive than the intersection test for original object. There is a trade of between complexity of expanded objects and the condition to have the smallest expansion possible (given by Lemma 3.1). Complexity of object gives us complexity of intersection test and smaller expansion causes our test to suceed in more cases.

## 4.4 Offsets versus Minkowski expansion

If the light source is a ball with given radius $d$ then the solid offset is the required expansion of the obstacle $Q$. With the notion of solid offsets, we obtain: a point $p$ is not in shade of an object $Q$ respective to a light source $B(c, d)$ if the ray $R(c, p)$ does not intersect the solid offset $\mathcal{O}_d(Q)$. Solid offsets are useful for several reasons. First, they are easy to evaluate for spheres since offsets of spheres are spheres with bigger radius. Second, for other simple geometric objects like cylinders and cones if we extend them by incresing their parameters the solid offset of the original object is included in such an extended object.. Third, there are specialized algorithms for calculating offsets of parametric surfaces which can be used.

There are situations where it is better to use solid offsets but Minkowski operators provide more effective algorithm for arbitrary shaped light sources. The advantage of Minkowski operators compared to solid offsets becomes clear when we look at non-spherical light sources.

**Example 4.1** *Let $L$ be a linear light source lying parallel to the axis $x$. Situation is illustrated in Figure 4.3. We want to test wheter the object $Q$ can obscure any shadow ray from the point $p$ to the light source $L$. We can see with the help of Minkowski operators that extended objects as required are bounded by the original bounding boxes extended only in dimension $x$ according to the size of the light source and a chosen central point $c$.*

*If we enclose a linear or planar light source in a bounding sphere, we can also contruct the test for penumbra region using solid offsetting. However, in that case the approximate shadow volume is unnecessary large. We have to expand all the bounding boxes of the objects in shadow scene equally in all directions.*

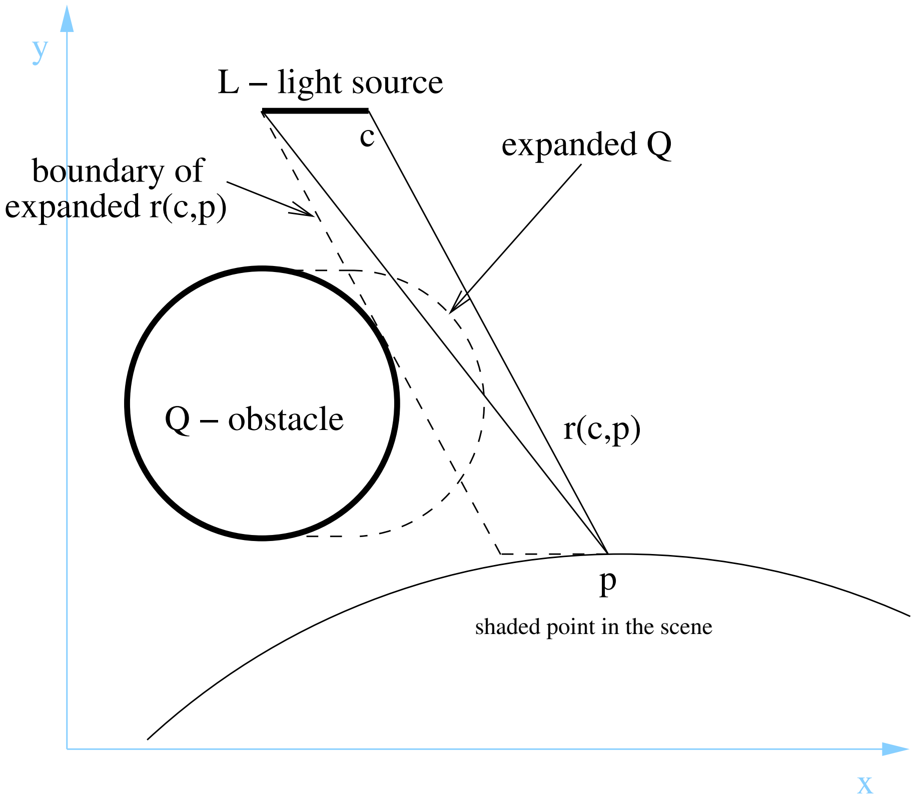

Figure 4.3: Linear light source causes smaller extended object.

## 4.5 Optimizations

Lemma 3.1 is a good criterion to detect possible shadow regions. However, there are points in full light which we do not detect. It follows from the fact that the visibility cone of rays from the light $L$ seen from point $p$ is smaller than the set of rays that meet criteria from the Lemma. We can further confine the possible shadow regions if we construct the shadow data set with smaller offsets (or with scaled sets and Minkowski operators). The idea is illustrated in Fig. 4.4. We start with spherical light sources and offsets and will present obvious generalizations.

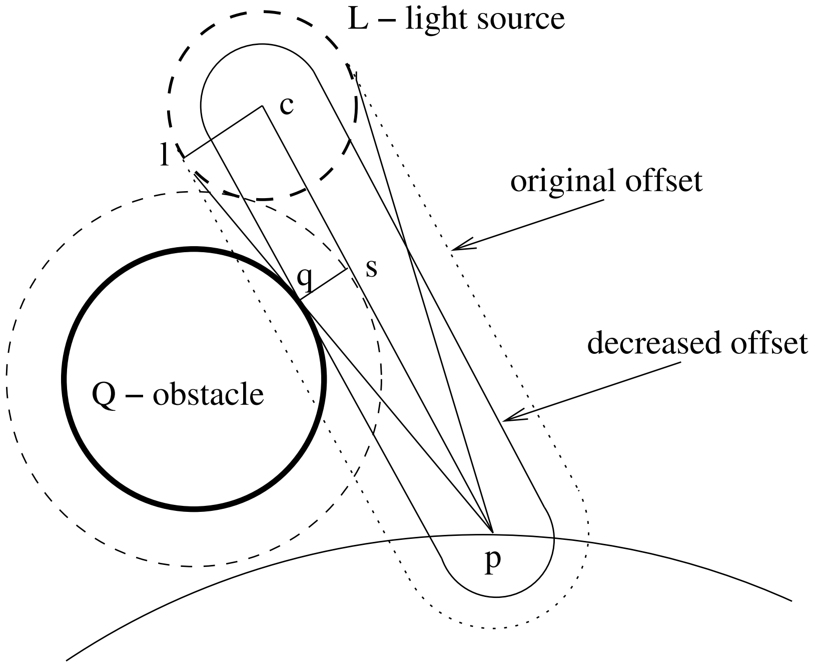

Figure 4.4: Radius optimization.

As we see in Fig. 4.4, instead of taking the solid offset of object $Q$ at distance $d(c,l)$ for a shadow intersection test, we can use the offset at distance $d(s,q)$ but for the test for these points. We can define in that case the ratio $t$ in which we can shrink the offset distance:

$$t := \frac{d(s,q)}{d(c,l)} \tag{4.1}$$

with simple geometry we can see that it is equivalent to:

$$t = \frac{d(s,q)}{d(c,l)} = \frac{d(p,q)}{d(p,l)} = \frac{d(p,q)}{d(p,q)+d(q,l)} \tag{4.2}$$

Since the operation of offsetting is global and should not depend on the points, we have to choose the maximum value of $t$ for all points $p \in S$, $l \in L$, and $q \in Q$. Using this ratio we can take smaller expansion still detecting the region containing both umbra and penumbra. We will define $t_{max}$ which would be valid for all possible points combinations.

$$\begin{aligned} t_{max} &= \max\left\{\frac{d(p,q)}{d(p,q)+d(q,l)} \;:\; p \in S,\; l \in L,\; q \in Q\right\} \leq \\ &\leq \frac{\max\{d(p,q):\; p \in S,\; q \in Q\}}{\max\{d(p,q):\; p \in S,\; q \in Q\} + \min\{d(q,l):\; q \in Q,\; l \in L\}} \end{aligned} \tag{4.3}$$

Using notation from Section 2.2 for the scaling of the set $A$ by a real number $t$ as $t \cdot A$, we can formulate the following result.

**Lemma 4.2** *Let $L$ be a star-convex set respective to a point $c \in L$ and let $t_{max}$ be defined by Equation 4.3 (for $S$, $L$ and $Q$). If the ray $R(c,p)$ does not intersect the set*

$$Q \ominus t_{max} \cdot (L \ominus \{c\}) \tag{4.4}$$

*then the point $p$ is not in shade of $Q$ respective to $L$.*

In the definition of $t_{max}$ the points $p$, $q$ belong to the scene, and the point $l$ is from the light source. Therefore we can calculate $t_{max}$ for the entire scene using the diameter of the scene (maximal distance between points $p$ and $q$) and the closest distance from the light source to the scene (minimal distance between points $q$ and $l$) . The improvement compared to Lemma 3.1 is significant if the distance to the light source is not too small compared to the diameter of the scene.

To get even better results we calculate $t_{max}$ for each pair of an obstacle and a light source and use it in expansion operation for the given obstacle. In the case of several light sources even if we use only one additional scenery for shadow rays it is also better to calculate separate $t_{max}$ values. Value of $t_{max}$ is a lower bound for the shrinking ratio and even if we do not compute this optimal shrinking ratio in each case, we can use any value $t$ with $1 > t \geq t_{max}$.

## 4.6 Umbra detection

The previous discussion allowed us to distinguish between the region in full light and the region in shadow. However, there might exist a totally shadowed umbra region where we need not to use stochastic ray tracing. We recall that the set complement is denoted as $\overline{A}$. In the following lemma we have the condition detecting umbra region.

**Lemma 4.3** *Let $L$ be a star-convex set with respect to a point $c \in L$. If the ray $R(c,p)$ intersects $\overline{\overline{Q} \ominus (L \ominus \{c\})}$ then a point $p$ is in the umbra of $Q$ with respect to the light source $L$.*

*Proof.* If the $R(c,p)$ intersects $\overline{\overline{Q} \ominus (L \ominus \{c\})}$ (see Fig. 4.5 for an illustration) then there exists a point $r \in R(c,p)$ such that $r \notin \overline{Q} \ominus (L \ominus \{c\})$. This is equivalent that for all $\bar{q} \notin Q$ and for all $l \in L$ holds $r \neq \bar{q} - (l - c)$. Using simple transformations, we obtain for all $\bar{q} \notin Q$ and for all $l \in L$ we have $\bar{q} \neq r + (l - c)$. This means that the set $\{r\} \oplus (L \ominus \{c\})$ is totally included in the set $Q$. But this is nothing else but the set $L$ translated by $r - c$. It is star–convex respective to the point $r$, hence for an arbitrary point $l \in L$ the ray $R(p,l)$ intersects it (the set $\{r\} \oplus (L \ominus \{c\})$ ). Since it is included in $Q$ the ray intersects object $Q$ what means that the point $p$ is totally shadowed and that $p$ is in the umbra region.

□

We can reformulate the condition given in Lemma 4.3 in the special case of spherical light sources using negative solid offsets which were defined in Section 2.2. The same optimization considerations for the distance as used in Section 4.5 apply, so we may use the bigger set as well:

$$\overline{\overline{Q} \ominus t_{max} \cdot (L \ominus \{c\})} \quad . \tag{4.5}$$

The umbra detection is done after having detected an object possibly casting shadow which is a good candidate to pass the test. Making the full test in a whole third data set holding only the shrunken objects requires more memory and more time. The gain is not sufficient since most of the tests fail. The experiments have shown that in almost all cases testing all the objects for umbra is slower than testing only one object which passed the penumbra detection test already.

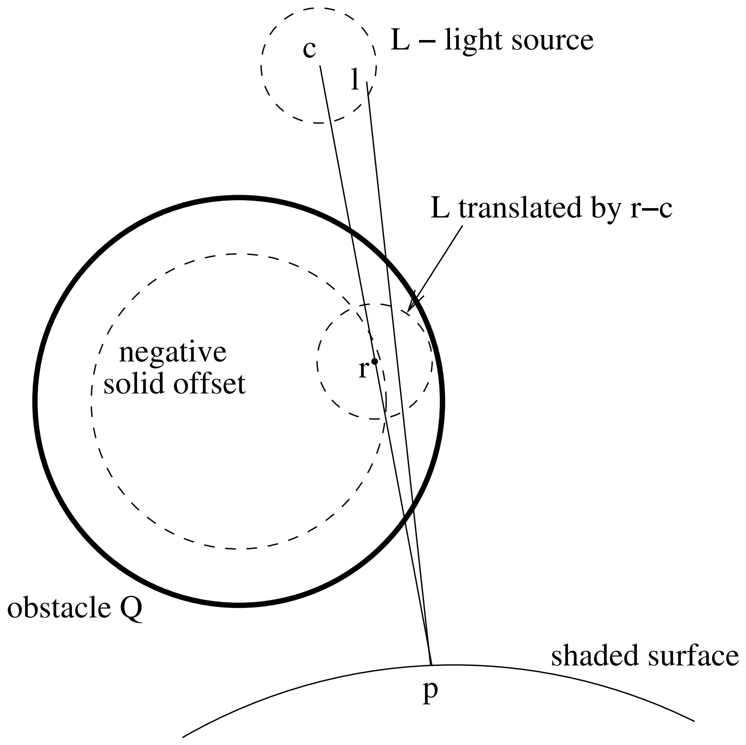

Figure 4.5: Umbra and negative solid offset.

## 4.7 Performance results

Our algorithm can be incorporated into any ray tracing kernel. Basically, it does not matter which classical acceleration method to enhance. Regular grids, octree, hierarchies of bounding boxes, or combinations of these can be used. We have incorporated our algorithm into RAYO ray tracer by A. Formella which is quite efficient. The ray tracer is based on an algorithm with modified BSP tree and plane traversal (for details see [12]).

| | | Cylinders | Balls4 | Rings | Molecule | Bust |
|---|---|---|---|---|---|---|
| *a)* | simple ray tracing | 0.91 | 5.42 | 1.83 | 1.34 | 3.15 |
| *b)* | stochastic ray tracing | 12.05 | 64.21 | 32.35 | 8.25 | 37.47 |
| | fast penumbra | | | | | |
| *c)* | standard | 6.10 | 43.48 | 15.11 | 5.66 | 32.46 |
| *d)* | optimized detection | 5.72 | 26.81 | 10.54 | 5.02 | 24.02 |
| *e)* | with umbra detection | 5.22 | 25.39 | 11.04 | 4.95 | |

Table 4.1: Run times in seconds for different images and algorithms.

**Remarks:** *a)* Traditional ray tracing which produces sharp shadows. *b)* Penumbra with classical stochastic ray tracing. *c)* Our method with shadow data sets. *d)* Optimized method with reduced extended objects according to the $t_{max}$ ratio. *e)* Additional umbra detection. (We still did not implement inner offsets for meshes.)

We compare the run times of the new method to the run times of traditional ray tracing without penumbra and classical stochastic ray tracing. More details are added as remarks in Table 4.1. We present several examples of geometric data sets: a simple scene with *cylinders* and spheres (Fig. 4.9), a complex *molecule* (Fig. 4.10) transformed from the Brookhaven Protein Data Bank, a *bust* (Fig. 4.11) modeled as a mesh of triangles, and the fractal *balls* (Fig. 4.12) and the *rings* (Fig. 4.13) from the SPD-benchmark [16]. The computations were done on a Sun SparcEnterprise 4000, 168 MHz, 1.125 GByte RAM and the presented datas reflect real time of the ray tracing loop without preprocessing. The test have been done being a single user so that the real time has been essentially equal to the user time.

Table 4.2 describes parameters of our test scenes together with the numbers of traced rays for images. The resolution for the tests has been set to $128 \times 128$ pixel. For larger images, the run times scale almost linearly with the resolution. The number $d$ of distributed rays was always set to 32. We enhanced the tracing of shadow rays with a shadow buffer. For each node of the ray tree a queue of up to two objects is buffered. A miss in the buffer enforces a delete of the last element in the queue, a hit of an object initiates an insertion as the first element of the queue.

In Table 4.2 for each method and each test scene we have the number of generated shadow rays and the average numbers of intersection tests per ray in geometric and shadow sceneries. The average numbers of intersections per ray with objects in both sceneries are quite low due to the well selected acceleration structures. The following trade-off can be observed. Our fast method significantly reduces the number of shadow rays to be traced. On the other hand, the number of intersection tests per ray is increased. The difference is larger if we use optimized decreased offsets. The umbra detection may not always pay-off, e. g., in the rings example, because the inner offset objects become too small. Additionally, we observed that the light cache hit rate has slightly improved for the more complex data sets.

Table 4.3 shows the memory requirements of the different implementations. As long as the scene description is small (simple scene) the additional memory required for the fast penumbra calculation is negligible. For the larger scenes at most twice as much memory is required.

The speedup which is obtained with the new method depends on the geometry, especially on the size of the light sources and on the size of the visible penumbras. Table 4.4 summarizes the run times for the balls3 data set with 822 objects. The size of the light sources was increased for the benchmark according to the sizes of the spheres being present in the data set. Stochastic ray tracing was performed with $d = 32$. The speedup for the fast method ranges from 1.76 to 7.83 depending on the size of the light sources: the smaller the light sources, the better the improvement in run time. The method well adapts to the size of the penumbra. If it is small there is less calculations and speed–up factor is bigger. If penumbra region is quite large the calculations are necessary anyway. Using only stochastic method the cost in both cases is the same i. e., unnecessary large.

In Fig. 4.6–4.8 we compare the run time of classical stochastic ray tracing with the run time of the version of the improved penumbra calculation (optimized decreased offsets according to Lemma 4.2). The figures show the dependance of run times on the number $d$ of stochastic rays per point. As it can be seen the dependance is close to linear for all scenes and for both methods. Comparing the slopes of the lines we obtain for large $d$ that the speedup for the simple scene is 2.1, for the molecule scene 2.7 and for the bust scene 1.8, respectively.

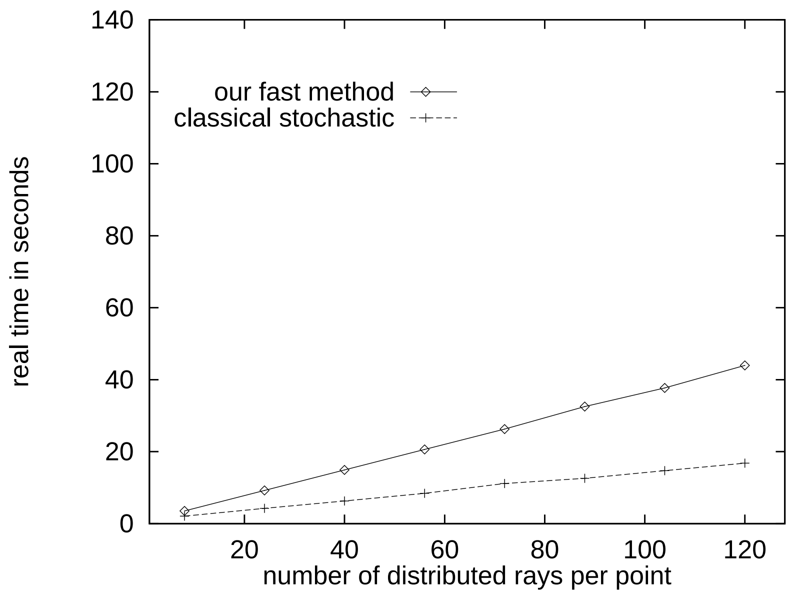

Figure 4.6: Run time dependance on the number of shadow rays per shaded point in the simple scene with cylinders.

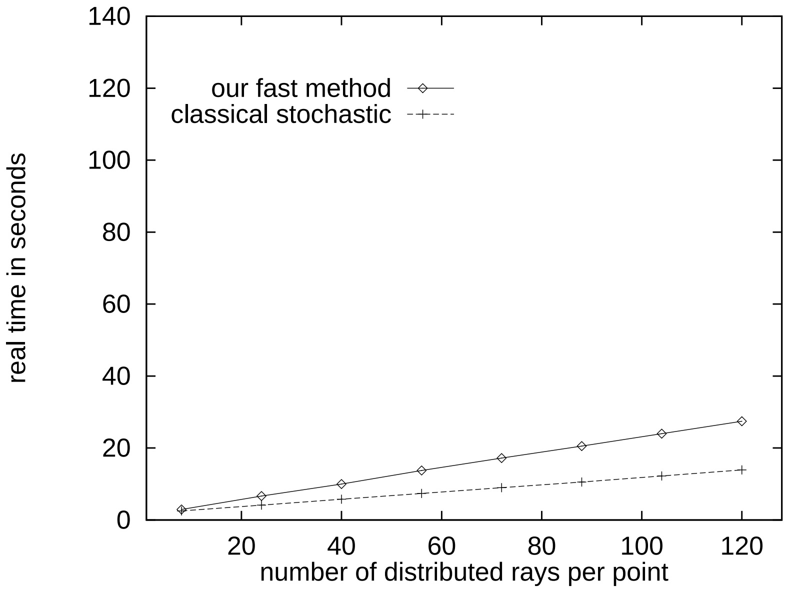

Figure 4.7: Run time dependance on the number of shadow rays per shaded point in the molecule scene.

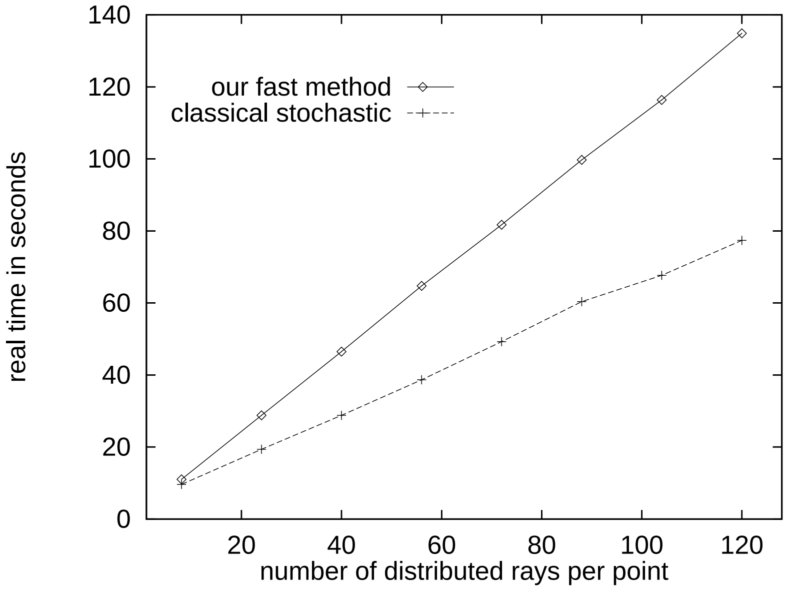

Figure 4.8: Run time dependance on the number of shadow rays per shaded point in the bust scene.

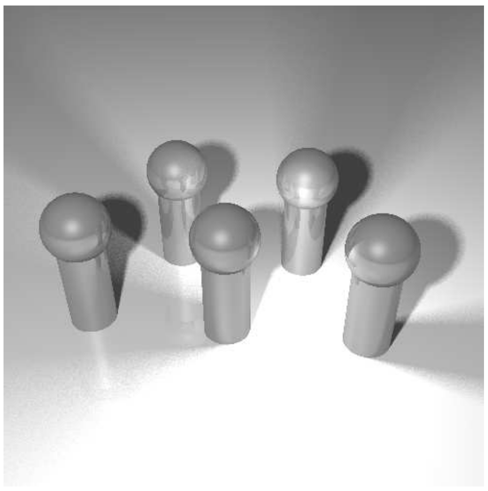

Figure 4.9: Penumbra in a simple scene.

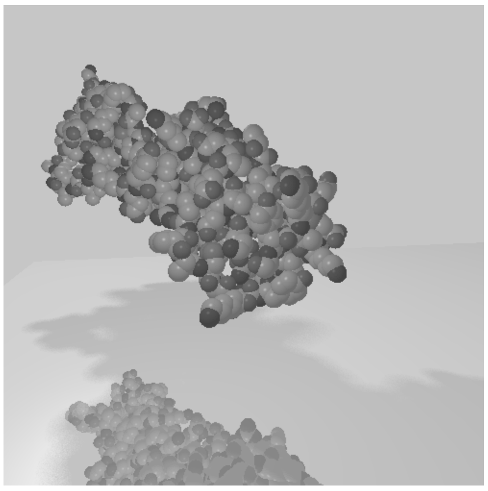

Figure 4.10: Penumbra for a compact molecule.

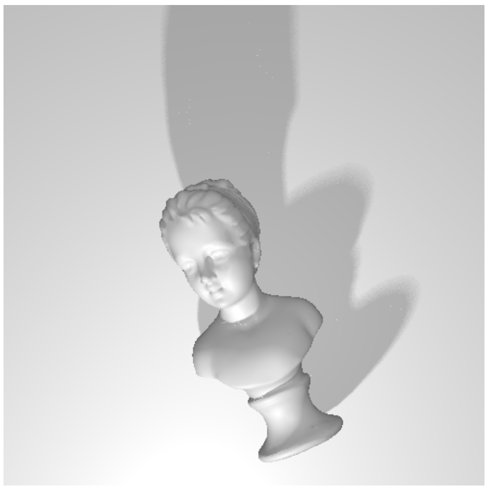

Figure 4.11: Penumbra in the bust scene.

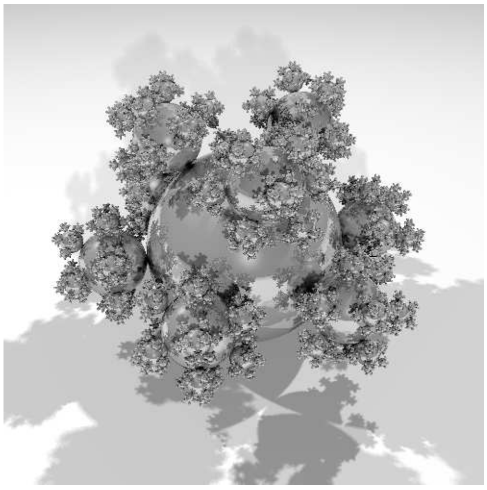

Figure 4.12: Penumbra in the balls scene.

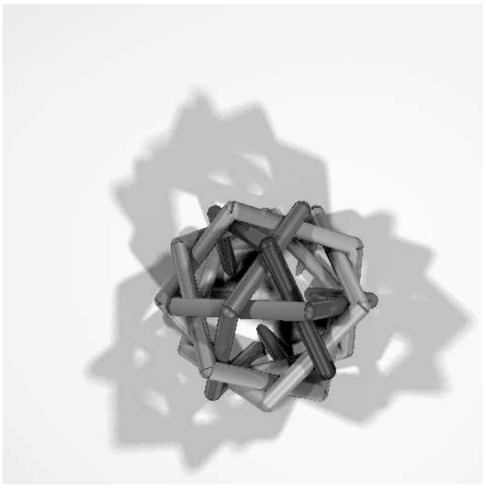

Figure 4.13: Penumbra in the rings scene.

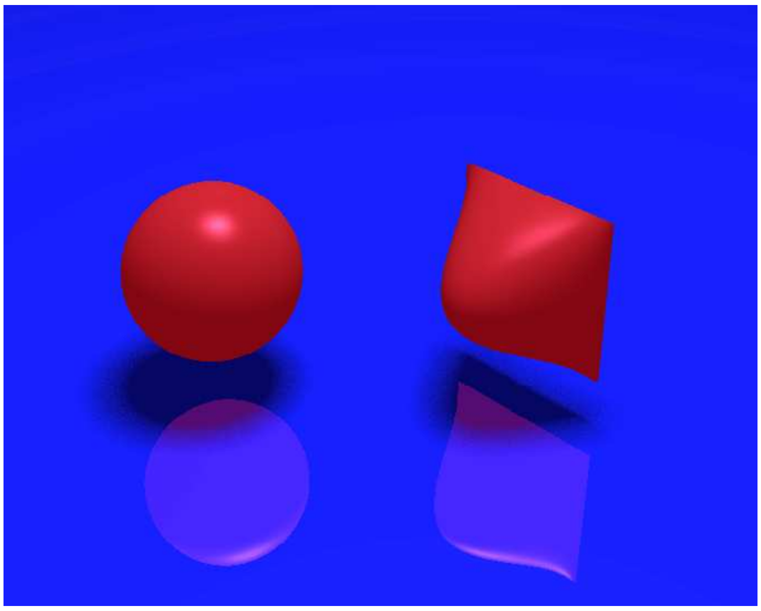

Figure 4.14: Penumbra in the scene with rational surface.

| | | Cylinders | Balls4 | Rings | Molecule | Bust |
|---|---|---|---|---|---|---|
| | # objects | 11 | 7383 | 62 | 1685 | 98506 |
| | # spherical light sources | 2 | 3 | 3 | 2 | 1 |
| | # reflected rays | 20167 | 10976 | 2962 | 7303 | 0 |
| *a)* | # shadow rays | 31592 | 51719 | 47415 | 4260 | 32136 |
| | # $I_{geom}$ | 0.85 | 1.74 | 1.40 | 0.99 | 0.94 |
| *b)* | # shadow rays | 1027757 | 1397529 | 1459345 | 105153 | 496704 |
| | # $I_{geom}$ | 1.30 | 2.36 | 1.22 | 1.80 | 2.13 |
| *c)* | # shadow rays | 324911 | 430300 | 367355 | 34615 | 196667 |
| | # $I_{geom}$ | 1.87 | 2.43 | 3.31 | 3.94 | 2.87 |
| | # $I_{shadow}$ | 1.13 | 2.32 | 1.72 | 2.27 | 4.62 |
| *d)* | # shadow rays | 289054 | 215151 | 215155 | 26800 | 135324 |
| | # $I_{geom}$ | 2.01 | 4.03 | 4.39 | 4.53 | 4.00 |
| | # $I_{shadow}$ | 1.08 | 0.85 | 1.39 | 1.79 | 2.05 |
| *e)* | # shadow rays | 287481 | 213120 | 215155 | 26691 | |
| | # $I_{geom}$ | 1.77 | 3.16 | 4.39 | 4.29 | |
| | # $I_{shadow}$ | 1.08 | 0.85 | 1.39 | 1.79 | |

Table 4.2: Characteristics of the example scenes.

**Remarks:** In all scenes the number of primary rays was equal to 16384. The numbers $I_{geom}$ and $I_{shadow}$ denote the number of intersection tests per ray in the geometry data set and in the shadow data set, respectively. For the description of the different methods see remarks in Table 4.1.

| | | Cylinders | Balls4 | Rings | Molecule | Bust |
|---|---|---|---|---|---|---|
| *b)* | simple ray tracing | 44 | 3064 | 82 | 670 | 58181 |
| *c)* | stochastic ray tracing | 45 | 3070 | 83 | 693 | 58181 |
| | fast penumbra | | | | | |
| *d)* | standard | 48 | 5535 | 114 | 1265 | 110176 |
| *d)* | optimized detection | 48 | 5991 | 117 | 1296 | 110197 |
| *e)* | with umbra detection | 49 | 5990 | 117 | 1296 | |

Table 4.3: Memory requirements in KByte for different images and algorithms.

**Remarks:** The numbers reflect dynamically allocated memory. In additional to the given values, a frame buffer of 50 KByte was allocated. For the description of the different methods see remarks in Table 4.1.

| | fast | class. | speedup |
|---|---|---|---|
| Balls3 | | 3.29 | |
| *a)* | 5.63 | 44.13 | 7.83 |
| *b)* | 9.88 | 44.33 | 4.48 |
| *c)* | 17.10 | 45.30 | 2.64 |
| *d)* | 27.21 | 48.10 | 1.76 |

Table 4.4: Run times for the Balls3 data set for different sizes of the light sources.

**Remarks:** The sizes of the light sources were set to the sizes of the spheres (*a)* smallest sphere, *d*) largest sphere). The first line shows the run time for simple ray tracing with no penumbra.

# Chapter 5

# Intersection methods

*"It can't go straight, you know, if you pin it all on one side,... "*

Lewis Carroll — Through The Looking Glass

We are interested in finding intersection of expanded objects with a ray. For rational surfaces it is not a trivial task. In our algorithm we will present these surfaces in Bézier form as they have been defined in Section 2.1. We will also use the notion of the Bézier volumes defined there. For a rational surface $S(u, v)$ of degree $(m, n)$ we want to find all intersections of its offset at given distance $d$ and a ray in parametric form $R(t)$.

Recall that $S_d(u, v)$ denotes an offset of a surface $S(u, v)$ at a distance $d$. The algorithm works as follows. Our goal is to solve in $\mathbb{R}^3$ the geometrical intersection problem:

$$S_d(u, v) = R(t) \tag{5.1}$$

In the first step we will transform the problem 5.1 into the problem of solving the following array of equations:

$$\begin{aligned} F(u, v, t) &= 0 \\ \partial_u F(u, v, t) &= 0 \\ \partial_v F(u, v, t) &= 0 \end{aligned} \tag{5.2}$$

where $F(u, v, t)$ will be defined in Corollary 5.3 as a polynomial of degree $(2m, 2n, 2)$. The solutions of the equation array 5.2 include all solutions of the Equation 5.1 but the reverse is not true. Formulations 5.1 and 5.2 are not equivalent.

Offset surface $S_d(u, v)$ in its definition contains the normal vector $N(u, v)$. The normal vector of a polynomial surface is not in general even rational function of the parameters $u$ and $v$ since to calculate it we use the square root (see Section 2.1.2). Therefore we significantly simplify the equations eliminating the square root which is not present in formulation 5.2.

In the second step we will solve the array of equations 5.2 by a subdivision method which is described in Section 5.2.2. The solutions of the system 5.2 can belong either to the positive offset $S_d(u, v)$ or to the negative one $S_{-d}(u, v)$. In the last step of the algorithm we will choose the right ones.

## 5.1 Main theorem

**Lemma 5.1** *Let $C(t)$ denote a parametric curve such that its derivative $\partial_t C(t)$ does not vanish in the neighborhood of the point $C(t_0)$, and let $P$ be a point such that $P \neq C(t_0)$. and let us define by the dot product the square distance function $f(t) :=< C(t) - P, C(t) - P >$. Then the vector $C(t_0) - P$ is orthogonal to the curve $C(t)$ at the point $C(t_0)$ iff $\partial_t f(t_0) = 0$.*

*Proof.* Using the following rule for calculating the derivative of the dot product:

$$\partial_t < A(t), B(t) >=< \partial_t A(t), B(t) > + < A(t), \partial_t B(t) >$$

we obtain:

$$\partial_t f(t_0) = \partial_t < C(t_0) - P, C(t_0) - P >= 2 < C(t_0) - P, \partial_t C(t_0) >$$

□

**Theorem 5.2** *Let $S(u, v)$ denote a regular parametric surface, let $R(t)$ be a curve and $d$ a positive real. Let the function $f(u, v, t)$ be defined by:*

$$f(u, v, t) := ||S(u, v) - R(t)||^2 - d^2$$

*Then the system of equations:*

$$\begin{aligned} f(u_0, v_0, t_0) &= 0 \quad (1) \\ \partial_u f(u_0, v_0, t_0) &= 0 \quad (2) \\ \partial_v f(u_0, v_0, t_0) &= 0 \quad (3) \end{aligned} \tag{5.3}$$

*is equivalent to the following alternative of two equations :*

$$R(t_0) = S_{-d}(u_0, v_0) \quad \text{or} \quad R(t_0) = S_d(u_0, v_0).$$

*Proof.* If $R(t_0) = S_i(u_0, v_0)$, for $i = -d$ or $i = +d$ then the distance from the point $R(t_0)$ to the point on the surface $S(u_0, v_0)$ is equal to $d$ and the first of Equations 5.3 holds. From the definition of the offset it follows that the vector $S(u_0, v_0) - R(t_0)$ is orthogonal to any curve on the surface at the point $S(u_0, v_0)$ and in particular to the curves $S(t, v_0)$ and $S(u_0, t)$. Lemma 5.1 applied to these curves gives the last two of Equations 5.3.

To prove the reverse implication assume that the last two of Equations 5.3 hold. By Lemma 5.1 the vector $S(u_0, v_0) - R(t_0)$ is orthogonal to linearly independent vectors $\partial_u S(u_0, v_0)$ and $\partial_v S(u_0, v_0)$. This means that the point $R(t_0)$ lies on the line orthogonal to the surface at the point $S(u_0, v_0)$. From the first of Equations 5.3 it follows that the distance of the points $R(t_0)$ and $S(u_0, v_0)$ is equal to $d$. Therefore $R(t_0) = S_{-d}(u_0, v_0)$ or $R(t_0) = S_{+d}(u_0, v_0)$.

□

**Corollary 5.3** *Let $S(u, v)$ be a regular rational Bézier surface in $\mathbb{R}^3$ defined by*

$$S(u, v) = \left( \frac{X(u, v)}{W(u, v)}, \frac{Y(u, v)}{W(u, v)}, \frac{Z(u, v)}{W(u, v)} \right)$$

*where $X(u, v), Y(u, v), Z(u, v), W(u, v)$ are polynomial functions such that $W(u, v) > 0$. Let $R(t) = (x(t), y(t), z(t))$ be a curve in $\mathbb{R}^3$, $d > 0$ a real number and let*

$$F(u, v, t) := (X - xW)^2 + (Y - yW)^2 + (Z - zW)^2 - d^2W^2$$

*Then the following system of equations:*

$$F(u_0, v_0, t_0) = 0 \tag{5.4}$$

$$\partial_u F(u_0, v_0, t_0) = 0 \tag{5.5}$$

$$\partial_v F(u_0, v_0, t_0) = 0 \tag{5.6}$$

*is equivalent to the alternative of two equations:*

$$R(t_0) = S_{-d}(u_0, v_0) \quad \text{or} \quad R(t_0) = S_d(u_0, v_0).$$

## 5.2 The Algorithm

Now we shall present a pseudocode of the algorithm based on the results given in Section 5.1.

**Input:**

A Surface $S(u,v)$ represented by a $(m,n)$-mesh of points in $\mathbb{R}^3$

An Offset distance $d$

A Parametric ray $R(t) = R_0 + tV$ where $R_0, V \in \mathbb{R}^3$

**Output:**

Set of parameters $(u,v,t)$ of intersection point

**The Algorithm:**

```
$F$ := CalculateVolumeFunction($S,d,R$);
PushVolume($F$);
**while** StackNotEmpty() **do begin**
  $F$ := PopVolume();
  (* Check if $0$ inside $F$, $\partial_u F$ and $\partial_v F$ *)
  **if** ZerosInsideHull($F$) **then**
    **if** (SmallEnough($F$)) **then**
      (* Generate and classify solution *)
      GenerateSolution($F$)
    **else begin**
      (* Split in one of directions $u,v,t$ sequentially *)
      SplitBezier($F$, $F_1$, $F_2$);
      PushVolume($F_1$);
      PushVolume($F_2$);
    **end**
**end**
```

### 5.2.1 Calculation of $F$

First we reparameterize the ray $R(t)$ by scaling the vector $V$ such that all solutions of our intersection problem 5.1 lie on the ray segment connecting $R_0$ and $R_0 + V$. Now for $R(t)$ we are interested in the parameter range $[0, 1]$.

In the line (1) of the algorithm we construct Bézier volume representaton for the function $F$ which was defined in Corollary 5.3. Control points of $F$ in the Bézier form can be calculated by simple arithmetic on control points which was defined in Section 2.1.1. Thus, we have Bézier volume $F$ defined by its control points and we can start the subdivision algorithm described below. In line (2) of the algorithm we push original Bézier volume on stack.

### 5.2.2 Subdivision and convex hull checking

Subdvision algorithm is a common way to solve intersection problems for Bézier curves and surfaces. It is based on de Casteljau subdivision algorithm (see Property 2.6) and convex hull property (see Property 2.7).

To solve the problem 5.2 we use the subdivision algorithm. We recursively split the Bézier volume each time into smaller volumes and using convex hull property we check if these small volumes can possibly contain solutions. That is we test if the convex hull of the set of control points of a Bézier volume $F_{ijk}$ contains the point $(0, 0, 0)$. We remove the volumes which can not contain solutions of our equation. We proceed until we run out of volumes which possibly contain solutions or until the split volume is small enough. In that second case we generate solution of the Equation 5.2.

We subdivide the Bézier volume $F$ sequentially in directions $u$, $v$ and $t$. To make each step of the the algorithm less expensive we use the convex hull property to check the "min–max" bounding box of $(F, \partial_u F, \partial_v F)$ instead of checking exact convex hull. That is we check each coordinate of the function $F(u, v, t)$ independently if convex hull of given coordinate of control points contains $0$. In our case we have to check if there are control points with different signs for the function $F$ and for its two partial derivatives.

By the Fact 2.12 the partial derivatives $\partial_u F$ and $\partial_v F$ of $F$ are easy to calculate using the control points of $F$. Therefore in our algorithm we calculate the control points of $\partial_u F$ and $\partial_v F$ online in each step when checking the convex hull property.

Thus we simplified the problem to the subdivision algorithm for the one function $F$ with modified convex hull checking.

The major cost of the subdivision algorithm is the cost of splitting the volume by de Casteljau algorithm. If we do not calculate exact convex hull then the time required for convex hull checking is neglible.

### 5.2.3 Classification of solutions

We are solving the array of equations 5.2 and from Corollary 5.3 we know that we get solutions for both offsets at distances $d$ and $-d$. If we want to consider only one offset $S_d(u, v)$ then we have to classify the solutions we have obtained. Since our solution contains all parameters which describe our situation completely we have what follows. The solution $(u, v, t)$ corresponds to an offset $S_d(u, v)$ if

$$Sign(d) = Sign(< N(u, v), R(t) - S(u, v) >).$$

## 5.3 Complexity analysis

The time for calulating the control points of Bézier volume $F$ can be neglected. Solving the equation system is the most time consuming part of the algorithm. The number of steps in the subdivision algorithm is a function of desired accuracy $\varepsilon$ and belongs to the class $O(\log(1/\varepsilon))$.

The cost of one step of the subdivision algorithm is the cost of the subdivision of the Bézier surface or volume using de Casteljau algorithm. We can represent this cost as the number of elementary operations of midpoint calculations. The surface is split into four subsurfaces and the volume is split into eight subvolumes. This guarantee that the parameter intervals will be smaller by half than before the subdivision. De Casteljau algorithm for curves of degree $n$ uses $n(n + 1)/2$ midpoint calculations.

In Table 5.1 we present the comparison of numbers of appropriate elementary operations for each subdivision step for our algorithm and others. All these algorithms use the subdivision algorithm but in each algorithm the reformulation of the problem is different therefore in each of them different objects are being subdivided.

**New Algorithm** is the algorithm presented here
— subdivides volume of degree $(2n, 2n, 2)$

**VP Polynomial** is the algorithm given by E. Vafiadou and N. Patrikalakis in [50]
— subdivides two surfaces of degree $(5n-2,\ 5n-2)$

**VP Rational** is a simple extension of **VP Polynomial** for the rational case
— subdivides two surfaces $(8n-2,\ 8n-2)$

| | Number of midpoint calculations | $n=2$ | $n=3$ | $n=4$ |
|---|---|---|---|---|
| New Algorithm | $6n^2(6n+4)$ | 384 | 1188 | 2688 |
| VP Polynomial | $(5n-2)^2(15n-3)$ | 1728 | 7098 | 18468 |
| VP Rational | $(8n-2)^2(24n-3)$ | 8820 | 33396 | 83700 |

Table 5.1: Numbers of midpoint calculations in each step of the subdivision algorithm for a surface of degree $(n, n)$.

As it can be seen from the Table 5.1, for the Bézier surfaces of degrees: 2, 3, 4 the subdivision phase for the new algorithm is 4–7 times faster for polynomial surfaces and 22–31 times faster for rational ones. Moreover for surfaces of higher degree the speed up is bigger.

## 5.4 Experimental results

The algorithm invented by Vafiadou and Patrikalakis [50] has been tested on a graphic workstation running at 20MHz. The results of these tests are given for the comparison purpose with our method in Table 5.2. We have implemented the new algorithm in "C" programming language.To measure the performance we included it in simple ray tracer which generated test rays. The tests of the new algorithm have been carried out on a slow 486DX2 66MHz microprocessor based computer running the *Linux* operating system. The effeciency of numerical calculations of this machine is comparable to the efficiency of 20MHz Sun workstations. We have also integrated the code into RAYO ray tracer written by A. Formella [12], which was used for generation of presented images.

The test results have been obtained for intersection with rays generated by a simple ray tracer implemented for this purpose. Thus for one surface many rays have been generated and the running times of the algorithm finding all intersections have been

| Surface description | Degree | VP results | New algorithm |
|---|---|---|---|
| Sweep of parabola | (1,2) | 2.7, 1.5 | 0.03 |
| Sweep of $x^4$ | (1,4) | 10.4, 6.7, 10.5 | 0.07 |

Table 5.2: Intersection times in seconds of the Vafiadou–Patrikalakis algorithm for few given test rays and average times of our new algorithm.

| Surface description | Degree | $d$ | Hit % | Aver. | Max. |
|---|---|---|---|---|---|
| Sweep of parabola | (1,2) | 0.5 | 31.3 | 0.03 | 0.16 |
| Sweep of $x^4$ | (1,4) | 0.5 | 30.0 | 0.07 | 0.38 |
| Rational Hill | (2,2) | 0.2 | 25.0 | 0.06 | 0.82 |

Table 5.3: Performance of the new algorithm (times in seconds).

obtained. Table 5.3 shows the average as well as the maximum running times in seconds (for accuracy of the algorithm set up to $0.0001$). The information how many rays hit the offset is also presented there since the algorithm runs faster when there is no intersection.

Following figures demonstrate images generated using the RAYO ray tracer and our intersection algorithm. The Figure 5.1 shows a sweep of parabola with two offsets at the distances $d = -0.5$ (below) and $d = 0.7$ (above). The Figure 5.2 shows two surfaces of degree $(2, 2)$ : the polynomial one (left) and the rational one (right) with the central weight stretched to $10$ (other control points have weight $1$). Next pictures show offsets of these surfaces with $d = -0.2$ (fig.5.3,5.4).

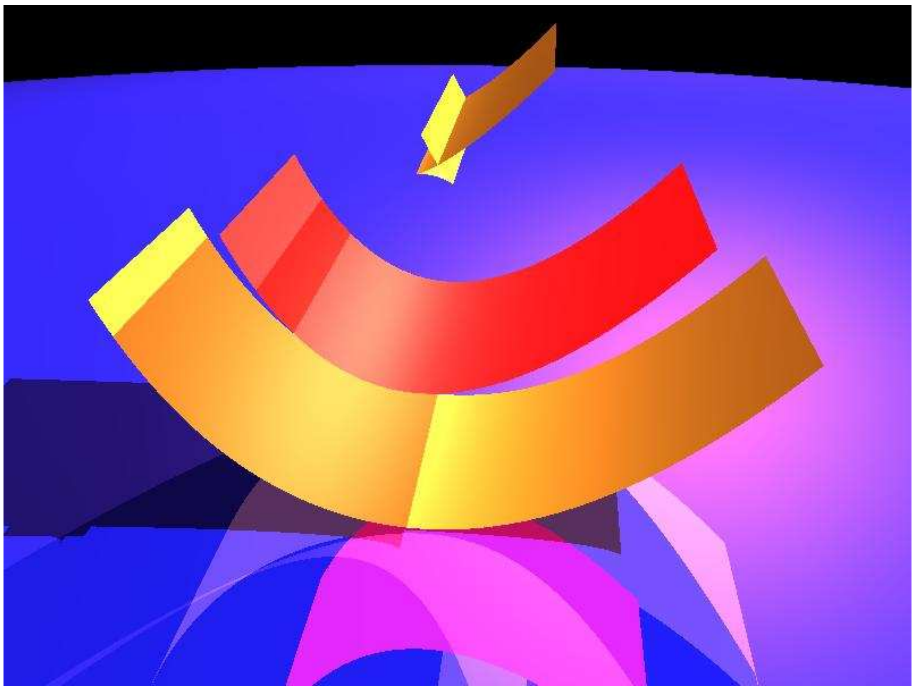

Figure 5.1: Offsets of sweep of parabola at $d = -0.5$ (below) and $d = 0.7$ (above)

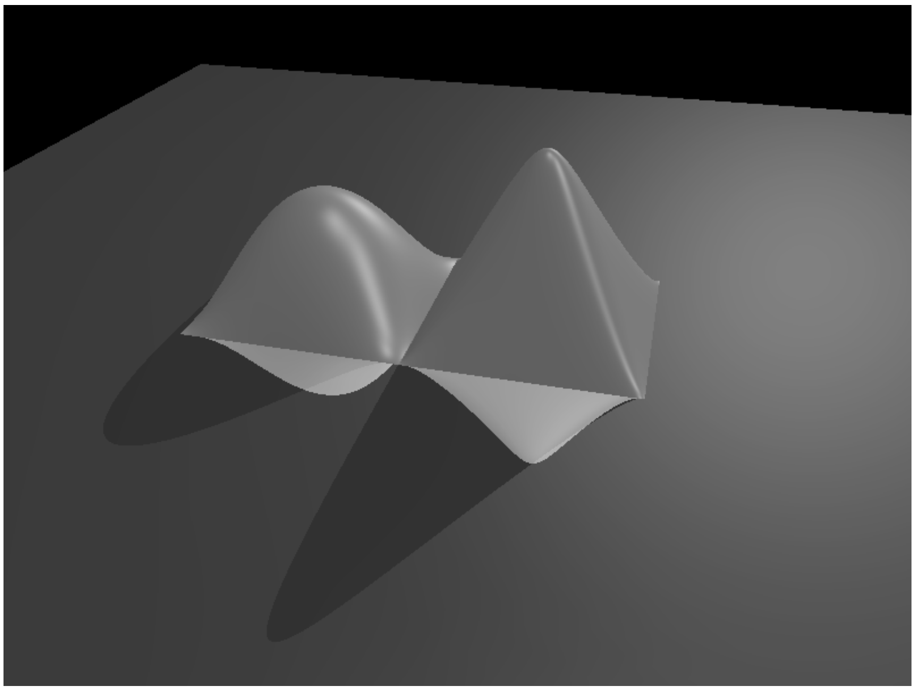

Figure 5.2: Polynomial and rational hills

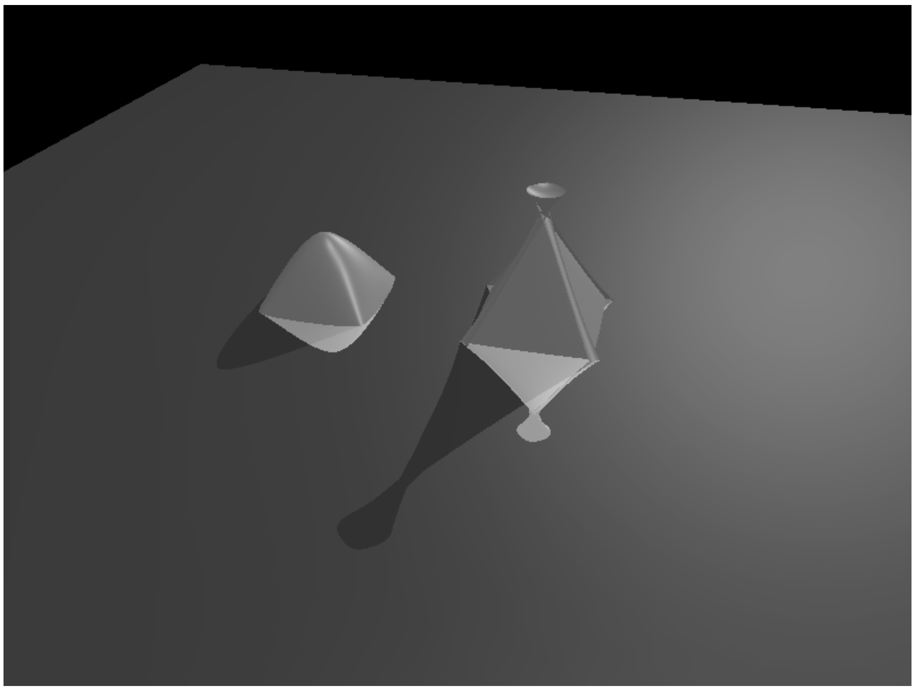

Figure 5.3: Offsets of polynomial and rational hills with $d = -0.2$

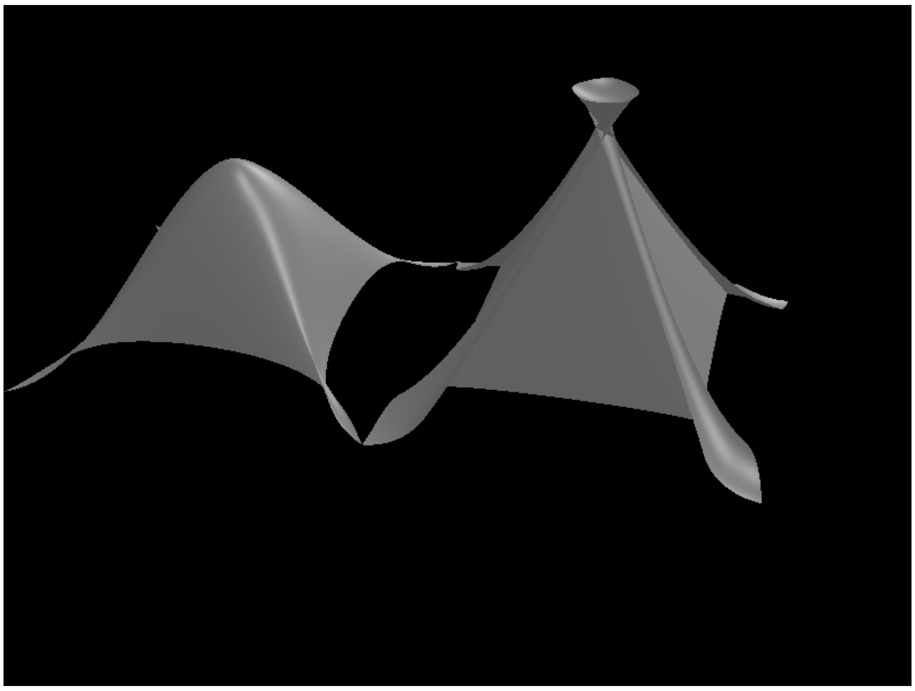

Figure 5.4: Offsets of polynomial and rational hills with $d = -0.2$

# Chapter 6

# Conclusions

*"That's all, said Humpty Dumpty."*

Lewis Carroll — Through The Looking Glass

We have presented a new method to speed up tracing of generalized shadow rays which is based on tracing the group of rays in the cost of tracing just one ray. The method is based on Minkowski operators and solid offsets. We have proved formally that if we use this method we will get the same results as without it. This guarantees the correctness of the results obtained.

We have constructed the a algorithm to speed up calculation of penumbra. The main idea is to detect the shadow regions such that stochastic ray tracing is confined to the penumbra. We have used the notion of Minkowski operators and solid offsets to provide the means to handle a variety of differently shaped light sources and objects. We have shown that the method works correctly and that it essentially renders the same images as stochastic ray tracing.

We have described and implemented an improvement of the basic algorithm which uses decreased offsets. The effectiveness of the algorithm depends on the particular geometric data set. On average, the presented sample scenes could be rendered two times faster compared to the run time of classical stochastic ray tracing. However, if the penumbra regions are small respective to the visible regions in the scene, much higher speedups can be obtained. The additional memory requirements never exceeded in our experiments a factor of two.

To show that the method works also for different kind of objects we have developed a new algorithm for rational Bézier surfaces. It finds all intersections of the ray segment with an offset of Bézier surface. The geometrical subdivision method and the Bézier reprezentation guarantee that the algorithm is robust (finds all solutions) and is numerically stable. The algorithm is substantially faster than the previously known one. Because of this, and because of advances in computing power of computers it is practical for photorealistic rendering of offset surfaces. As an example, the ray traced images of cubic rational surfaces have been presented (average intersection time in this case was less than a milisecond on a fast workstation). The algorithm can be also applied to collision detection and to other areas in CAGD, motion planning and machine milling.

The general method of speeding up shadow ray tracing seems to be very appropriate to be incorporated into any ray tracing system. To exploit further possibilities, cases where the rays are not given in coherent groups should be examined. However, the memory requirements are getting bigger in that case, and there are additinal costs for accesing special ray caching structures which have to be constructed.

As we have shown in our experiments for new algorithm in Chapter 4 adding umbra detection did not improved significantly the performance. However, performance depends on the test scenes. We are not excluding that it will be profitable in some special cases when the umbra regions are large. Further research might also investigate the calculation of combined convex *bounded* volumes that would allow detection of the umbra more precisely (for instance for CSG-models). A single bounded volume might be calculated for a number of joined objects.

For complex data set, we expect a further improvement in rendering time of new algorithm from Chapter 4, if an additional space subdivision is employed for the candidate objects casting penumbra. If the test for a ray in shadow scene fails, we can have the list of objects which can potentially obstruct the group of rays. Therefore if we use some acceleration structures for this list of candidate objects it will be faster to trace the rays from this group in this small subset of objects.

# Bibliography

[1] Maneesh Agrawala, Ravi Ramamoorthi, Alan Heirich, and Laurent Moll. Efficient image-based methods for rendering soft shadows. *Proceedings of SIGGRAPH 2000*, pages 375–384, July 2000.

[2] John Amanatides. Ray tracing with cones. *Computer Graphics (SIGGRAPH '84 Proceedings)*, 18(3):129–135, July 1984.

[3] Hujun Bao and Qunsheng Peng. Shading models for linear and area light sources. *Computers and Graphics*, 17(2):137–145, 1993.

[4] Wolfgang Boehm, Gerald Farin, and Jurgen Kahmann. A survey of curve and surface methods in cagd. *Computer Aided Geometric Design*, 1(1):1–60, 1984.

[5] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. *Computer Graphics (Proceedings of SIGGRAPH 84)*, 18(3):137–145, July 1984.

[6] Gerald Farin. Curvature continuity and offsets for piecewise conics. *ACM Transactions on Graphics*, 8(2):89–99, 1989.

[7] Gerald Farin. Curves and Surfaces for Computer Aided Geometric Design. Academic Press, 1990.

[8] Rida T. Farouki. Exact offset procedures for simple solids. *Computer Aided Geometric Design*, 2(4):257–279, 1985.

[9] Rida T. Farouki. The approximation of non-degenerate offset surfaces. *Computer Aided Geometric Design*, 3(1):15–43, 1986.

[10] Rida T. Farouki and C. Neff. Algebraic properties of plane offset curves. *Computer Aided Geometric Design*, 7:101–128, 1990.

[11] Rida T. Farouki and C. Neff. Analytic properties of plane offset curves. *Computer Aided Geometric Design*, 7:83–100, 1990.

[12] Arno Formella and Christian Gill. Ray Tracing: A Quantitative Analysis and a New Practical Algorithm. *The Visual Computer*, 11(9):465–476, December 1995.

[13] Akira Fujimoto, Takayuki Tanaka, and Kansei Iwata. ARTS: Accelerated ray tracing system. *IEEE Computer Graphics and Applications*, 6(4):16–26, April 1986.

[14] Andrew S. Glassner (editor). An Introduction to Ray Tracing. Academic Press, 1989.

[15] Andrew S. Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, 4(10):15–22, October 1984.

[16] Eric A. Haines. A proposal for standard graphics environments. *IEEE Computer Graphics and Applications*, 7(11):3–5, November 1987.

[17] Eric A. Haines and Donald P. Greenberg. The light buffer: A ray tracer shadow testing accelerator. *IEEE Computer Graphics and Applications*, 6(9):6–16, September 1986.

[18] Vlastimil Havran. Heuristic Ray Shooting Algorithms. Ph.d. thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, November 2000.

[19] Wolfgang Heidrich, Stefan Brabec, and Hans-Peter Seidel. Soft shadow maps for linear lights. *Rendering Techniques 2000: 11th Eurographics Workshop on Rendering*, pages 269–280, Springer–Verlag Wien New York, June 2000.

[20] Paul S. Heckbert and Pat Hanrahan. Beam tracing polygonal objects. *Computer Graphics (SIGGRAPH '84 Proceedings)*, 18(3):119–127, July 1984.

[21] Paul S. Heckbert and Michael Herf. Simulating soft shadows with graphics hardware. Technical Report CMU-CS-97-104, CS Dept., Carnegie Mellon U., January 1997. http://www.cs.cmu.edu/ ph.

[22] Joseph Hoschek and Dieter Lasser. Fundamentals of Computer Aided Geometric Design. A K Peters, 1993.

[23] Joseph Hoschek. Spline approximation of offset curves. *Computer Aided Geometric Design*, 5(1):33–40, 1988.

[24] Henrik W. Jensen and Niels J. Christensen. Efficiently rendering shadows using the photon map. *Compugraphics '95*, pages 285–291, December 1995.

[25] Henrik Wann Jensen. Global illumination using photon maps. *Eurographics Rendering Workshop 1996*, pages 21–30, Springer–Verlag Wien New York, June 1996.

[26] James T. Kajiya. The rendering equation. *Computer Graphics (Proceedings of SIGGRAPH 86)*, 20(4):143–150, August 1986.

[27] M. Kaplan. Space-tracing: A constant time ray-tracer. *SIGGRAPH '85 State of the Art in Image Synthesis seminar notes*, 18(3):149–158, July 1985.

[28] Brett Keating and Nelson Max. Shadow penumbras for complex objects by depth-dependent filtering of multi-layer depth images. *Eurographics Rendering Workshop 1999*, pages 197-212, Springer–Verlag Wien New York, June 1999.

[29] J.-C. Latombe. Robot Motion Planning. Kluwer Academic Publishers, 1991.

[30] G.W. Leibniz. Generalia de natura linearum anguloque contactus et osculi provocationibus aliisque cognatis et eorum usibus nonnullis. *Acta Eruditorum*, 1692.

[31] Mark E. Lee, Richard A. Redner, and Samuel P. Uselton. Statistically optimized sampling for distributed ray tracing. *Computer Graphics (SIGGRAPH '85 Proceedings)*, 19(3):61–67, July 1985.

[32] Eric P. Lafortune and Yves D. Willems. Bi–directional path tracing. *Proc. of CompuGraphicsm (Alvor, Portugal)*, pages 145–153, 1993.

[33] Eric P. Lafortune and Yves D. Willems. Reducing the number of shadow rays in bidirectional path tracing. *WSCG'95 Conference Proceedings,* pages 384-392, University of West Bohemia, February 1995.

[34] Andrzej Łukaszewski. Exploiting Coherence of Shadow Rays. *AFRIGRAPH 2001 Conference Proceedings* (to be published by ACM SIGGRAPH).

[35] Andrzej Łukaszewski, Andrzej Szczepkowicz. Computer simulation of FIM images — the convex hull model. *Vacuum (Elsevier Science Ltd),* 54(1999), pages 67-71.

[36] Andrzej Łukaszewski and Arno Formella. Fast penumbra calculation in ray tracing. *WSCG'98 Conference Proceedings,* Vol. II, pages 238–245, University of West Bohemia, February 1998.

[37] Andrzej Łukaszewski. Finding ray-offset intersection for rational Bézier surfaces. Technical Report 97/04, Institute of Computer Science, University of Wrocław, Poland, May 1997.

[38] Andrzej Łukaszewski. Evolutionary Programming In Graph Coloring. *Badania Operacyjne i Decyzje,* Nr. 3/1995 pages 67-74.

[39] Masataka Ohta and Mamoru Maekawa. Ray coherence theorem and constant time ray tracing algorithm. *Computer Graphics 1987 (Proceedings of CG International '87),* pages 303–314. Springer–Verlag, 1987.

[40] Steven Parker, Peter Shirley, and Brian Smits. Single sample soft shadows. Technical Report UUCS-98-019, Computer Science Department, University of Utah, October 1998. http://www.cs.utah.edu/~bes/papers/coneShadow.

[41] Sumant N. Pattanaik. Computational methods for global illumination and visualisation of complex 3d environments. published by Birla Institute of Technology & Science, Computer Science Department, February 1993.

[42] Sumant N. Pattanaik and S. P. Mudur. Computation of global illumination by monte carlo simulation of the particle model of light. *Third Eurographics Workshop on Rendering,* pages 71–83, May 1992.

[43] Sumant N. Pattanaik and S. P. Mudur. The potential equation and importance in illumination computations. *Computer Graphics Forum,* 12(2):131–136, 1993.

[44] Andrew Pearce and David Jevans. Exploiting shadow coherence in ray tracing. *Proceedings of Graphics Interface '91,* pages 109–116, June 1991.

[45] Bui-T. Phong. Illumination for computer generated pictures. *Communications of the ACM,* 18(6):311–317, June 1975.

[46] Helmut Pottmann. Rational curves and surfaces with rational offsets. *Computer Aided Geometric Design*, 12:175–192, 1995.

[47] Pierre Poulin and John Amanatides. Shading and shadowing with linear light sources. *Eurographics '90*, pages 377–386. North–Holland, September 1990.

[48] Jerek. Rossignac and A.A.G. Requicha. Offseting operations in solid modelling. *Computer Aided Geometric Design*, 3:129–148, 1986.

[49] B. Silverman. Density Estimation for Statistics and Data Analysis. Chapman and Hall, Ltd. London, 1985.

[50] Maria-E. Vafiadou and Nicholas M. Patrikalakis. Interrogation of offsets of polynomial surface patches. *Eurographics '91*, pages 247–259. North–Holland, September 1991.

[51] Eric Veach. Robust Monte Carlo Methods For Light Transport Simulation. *Ph.d. thesis,* Stanford University, 1997.

[52] Ingo Wald, Philipp Slusallek, Carsten Benthin, and Markus Wagner. Interactive rendering with coherent ray tracing. *Computer Graphics Forum*, 20(3), 2001.

[53] Leonard Wanger. The effect of shadow quality on the perception of spatial relationships in computer generated imagery. *Computer Graphics (1992 Symposium on Interactive 3D Graphics)*, 25(2):39–42, March 1992.

[54] Turner Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349, June 1980.

[55] Andrew Woo. Efficient shadow computations in ray tracing. *IEEE Computer Graphics and Applications*, 13(5):78–83, September 1993.

[56] Andrew Woo, Pierre Poulin, and Alain Fournier. A survey of shadow algorithms. *IEEE Computer Graphics and Applications*, 10(6):13–32, November 1990.