Internship: CPU Image Based Global Illumination

Matthias Labschütz, 8971103

August 9, 2014

Contents

Chapter 1

Algorithm Requirements

The the approach calculates first and second bounces in global illumination using rasterization with summation over the pixels of rasterized image planes. The surfaces are perfectly diffuse reflectors.

1.1 Direct Illumination

The direct illumination (first bounce) is calculated similar to rasterization in OpenGL. The following formula shows the direct illumination intensity I at pixel position p as:

$$I_p^{direct} = \sum_{l=0}^{Ls} I_l \frac{1}{1 + |(p_{pos} - l_{pos})|^{exp}} f(N_p, L_{lp}) \tag{1.1}$$

Ls is the number of light-sources. I_l is the intensity and color of the light-source l.

$\frac{1}{1+|(p_{pos}-l_{pos})|^{exp}}$ is the distance fall-off, with p_{pos} and l_{pos} as the world space positions of the pixel p and the light-source l and the fall-off exponent exp. $exp = 2$ is a quadratic fall-off and $exp = 1$ a linear fall-off.

f is the dot product for perfectly diffuse surfaces, as defined in 1.3. N_p is the surface normal at pixel position p and L_{lp} the normalized direction vector from the light-source l to the pixel p.

1.2 Indirect Illumination

For the second bounce, the following calculation is necessary:

$$I_p^{total} = I_p^{direct} + \sum_{l=0}^{Ls} \sum_{pc=0}^{Pc_l} I_{pc}\, f(N_p, L_{pc}^p)\, \frac{1}{1 + |(pc_{pos} - p_{pos})|^{exp}}\, g(Nc_p, L_p^{pc})\, nc\, pc_{color} \tag{1.2}$$

Pc_l are all pixels rasterized from the perspective of the light-source l. L_{pc}^p is the normalized light direction vector from pc to p and L_p^{pc} from p to pc (the purple line segment in figure 1.1). nc is a normalization constant that depends on the resolution of the image planes. pc_{color} is the color of the pixel.

For perfectly diffuse surfaces the function g is defined as:

$$f(x, y) = g(x, y) = \begin{cases} 0 & \text{if } x \cdot y < 0 \\ x \cdot y & \text{else} \end{cases} \tag{1.3}$$

where $x \cdot y$ is the dot product between two vectors x and y.

Figure 1.1 shows that as a result of g, the pixel p is not light by the pixel pc since the surface normal faces away from the pixel p. It is possible to define other functions g for the reflections on the surfaces.

The sum over Ls is done for all light-source image planes, the sum Pc for all pixels of a lightsource image plane.

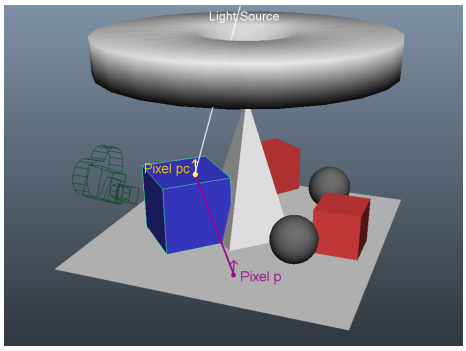


Figure 1.1: Indirect illumination (second bounce) at pixel p. The light reflected from pixel pc does not influence pixel p. In this scene, no blue color bleeding occurs on the white ground plane for the first and second bounce.

Equation 1.2 can be rewritten to:

$$I_p^{total} = I_p^{direct} + \sum_{l=0}^{Ls} nc \sum_{pc=0}^{Pc_l} I_{pc}\, f(N_p, L_{pc}^p)\, \frac{1}{1 + |(pc_{pos} - p_{pos})|^{exp}}\, g(Nc_p, L_p^{pc})\, pc_{color} \tag{1.4}$$

If one of the factors I_{pc}, $(N_p \cdot L_{pc}^p)$, $\frac{1}{1+|(pc_{pos}-p_{pos})|^{exp}}$ or $f(Nc_p, L_p^{pc})$ is zero, the summation part of this pixel value is zero.

The goal in the following is to find optimizations that catch a zero value for either of these factors on a coarser level than on the per pixel sum. For example, if all intensities of a light-source are zero, the whole sum would be zero and it would not be necessary to sum over it at all.

It can be seen that the light direction L changes for every summation step, while the normal is constant for a single pixel.

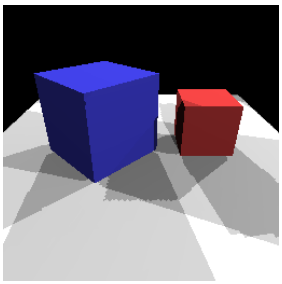
The distance between pc_{pos} and p_{pos} varies between every pair of pixels.

1.2.1 Implementation

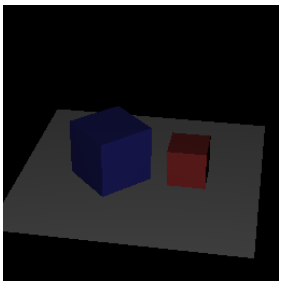
The algorithm works in the following way:

1. Render the final image plane in three different view-ports containing. The albedo color, the normals and the depth value.
2. Render an image plane for every light-source (similar as in shadow mapping). For each image plane the albedo color, the normal, the depth and the light intensity are stored. These images are shown in figure 1.2b to 1.2f.
3. Calculate the direct illumination by projecting the light-source image planes onto the final image plane. Like this, the direct illumination of the pixels, as well as the shadow regions are rendered. The result is shown in figure 1.2a.
4. For every pixel, calculate the indirect illumination for an additional bounce. The final result is shown in 1.2g.

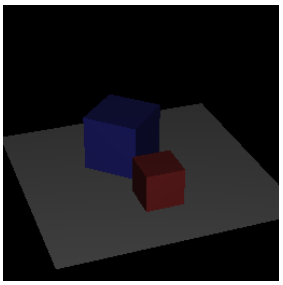
Step four was explained in formula 1.2. The algorithm sums for every pixel over all light-source image pixels. In this summation, the illumination is calculated according to the distance and direction of every pixel in the light-source images with respect to the final image pixels.



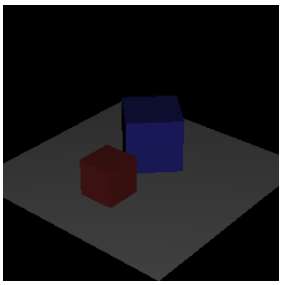
(a) Direct Illumination



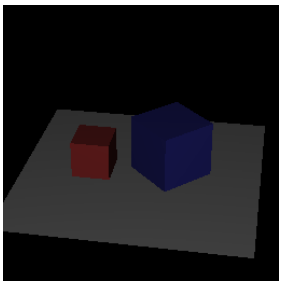
(b) First light-source



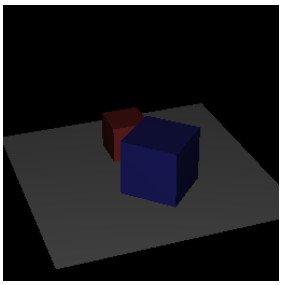
(c) Second light-source



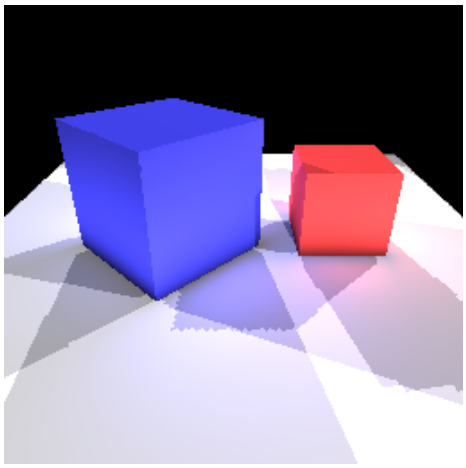
(d) Third light-source



(e) Fourth light-source



(f) Fifth light-source



(g) Final image

Figure 1.2: Different steps during the algorithm

Chapter 2

Algorithm Improvements

The goal of a lossless performance optimization is to reduce the summation for parts that do not require to be summed. These are the parts that are zero.

The summation has to be done for every pixel of the resulting image. Therefore, if it is possible to generate a data-structure that simplifies the summation, each of the $size_x * size_y$ steps can potentially be speed up. This means, that even a more complex data-structure may pay off, since it needs only to be generated once and can be used throughout all summation steps.

The following cases result have zero impact on the summation:

- Intensities of the image are zero.
- Distance between two points (or areas) are too large.
- Normals are orthogonal to each other or pointing away from each other.

There are possibly two ways to tackle the problem. One way is to only consider a coarser look-up structure (e.g. quad-tree) for every light-source image. The other way would be to coarsen the look-ups also in the result image. For example to find batches of non-illuminated pixels in the final image faster.

2.1 Light-Source Image Space Simplification

In the following, only the approach of a reduction on a per light-source level is discussed.

2.1.1 Intensity Discard

Discarding intensities that are zero can be done by generating a quad-tree on the intensity image. The sum over all pixels in Pc can happen on a more coarse level of the quad tree. For every summation step the intensity I_{pc} maximum is retrieved. If the maximum is below zero the whole sub-tree can be skipped.

Note: The summation is achieved by a recursion through the tree. Additionally a method was implemented that allows leafs with more than one element (e.g. four leaf pixels). A summation in this case requires a sum of all leaf elements within the leaf.

2.1.2 Distance Discard

Discarding distances that are too large do not need to be considered in the illumination summation. Therefore it is possible to generate a quad-tree that tests for the maximal require a bounding structure to be saved on a per-quad-tree level. Since illumination due to a surface con only occur in direction of the normal, only half-spheres are needed. Since the smallest enclosing space of half-spheres is hard to find, only bounding-spheres were used.

It is possible to use bounding spheres, therefore the quad-tree has to save a center position and a radius for every level. Another option would be to use axis aligned bounding-boxes. The advantage of bounding-boxes is, that the generation of the quad tree is easier (only store minimum and maximum). The disadvantage is that the distance calculation is more complicated.

Purpose of the data-structure

For every pixel p a look-up in the bounding sphere quad-tree data structure is made. At every node the data structure stores the centroid c and the radius r. If p is inside the bounding sphere it is illuminated, if not, it can be discarded.

Generation of the data-structure

The generation of the quad-tree is non-trivial and needs to do the following: Store the maximum distance (centroid and radius) at which an intensity inside the current leaf region still has enough intensity to illuminate a point.

For finding the smallest enclosing ball of points (bounding-sphere of points), Welzls algorithm can be used [Wel91]. In this case, the smallest enclosing ball of balls is required, this problem can not be solved as efficiently as described in [FG03].

As a result the following heuristic was used:

- Calculate the bounding-box for all spheres contained in the quad-tree node. A sphere object consists of center point and radius.
- Use the bounding-box center as center and diagonal as radius.

The adding of spheres to the quad-tree nodes was done recursively, but the calculation of bounding-spheres does not re-use child information.

2.1.3 Normal Discard

The normal discard approach was not implemented, and just serves as a further idea to speed up the algorithm.

Average normal deviation

Also normals can be discarded if the corresponding surfaces face away from each other.

An idea to formulate an interval of normals at a quad-tree leaf would be to take the average normal (centroid) and its maximum deviation. This does not work if the normals lie outside a common half-sphere. Since we render the scene from a light-source perspective, with a small enough field of view angle, the normals visible in a view-port are always in a common half-sphere.

The following approach would work for generating a quad-tree for the normal: At each leaf, generate the average normal and its maximum deviation angle and store it. Notice, this can not be done incrementally, the average normal and the maximum deviation angle has to be calculate from all candidates. But there may be approaches to improve it.

During rendering, it is necessary to test the angle of the angle of the current normal to the average normal, against the maximum deviation angle.

Candidate normal deviation

This approach would test the normal to a candidate solution. For example with respect to the 3 basis vectors of a Cartesian coordinate system (and its 3 negated vectors).

2.2 Light-source and Camera Image Space Simplification

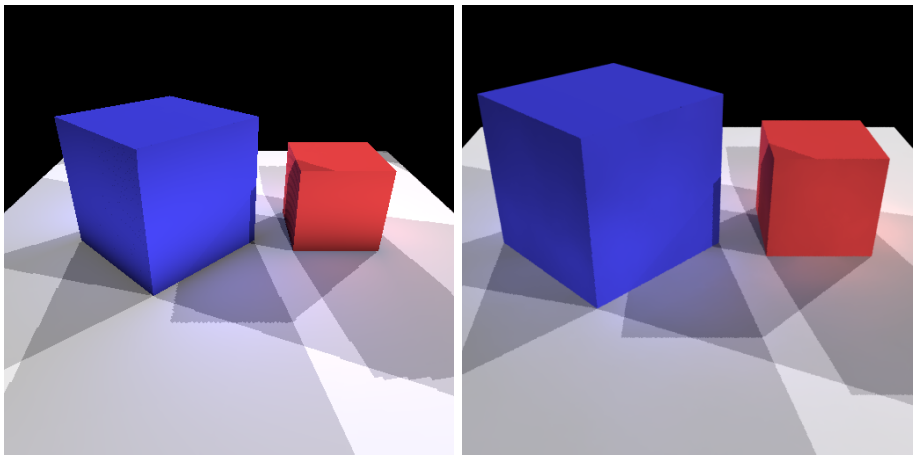
It is also possible to generate a quad-tree for the camera pixels itself. This makes sense if either the part of the screen is unlit, or if it is of interest to create a lossy representation of the final result.

2.2.1 Distance Discard

If a pixel is too far away from the light-sources, also its neighbor pixels might be too far away from the light sources. In specific, this would require to create a bounding-sphere hierarchy with a quad-tree and test a sphere (not the point) with the sphere of maximum light influence as described in 2.1.2.

2.3 Results

Three different scenes were rendered. For comparison a Mental Ray Photon mapping was used.



(a) Image based global illumination result. (b) Mental Ray Photon mapping result.

Figure 2.1: Rendering of the first scene with $nc = \frac{|P_c|}{200}$

In figure 2.1a the first image was rendered with the distance discard approach. It contains five light-light-sources. In comparison to the Photon mapping approach in 2.1b, an anti-aliasing problem is visible. This is especially visible at the shadow borders. Higher resolution shadow-maps did not improve the result. It is suspected, that an interpolation method for accessing image plane data could improve the shadow quality.

In addition, the border of the cube towards the ground plane shows ambient occlusion like artifacts. This is caused by the nature of Lambert's cosine law as shown in figure 2.2.

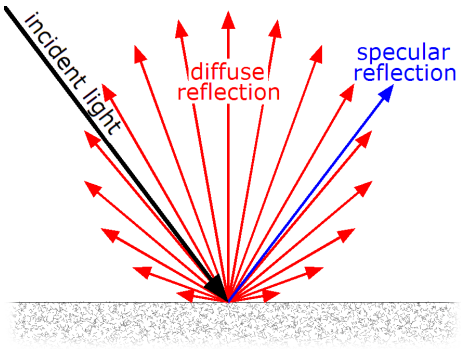
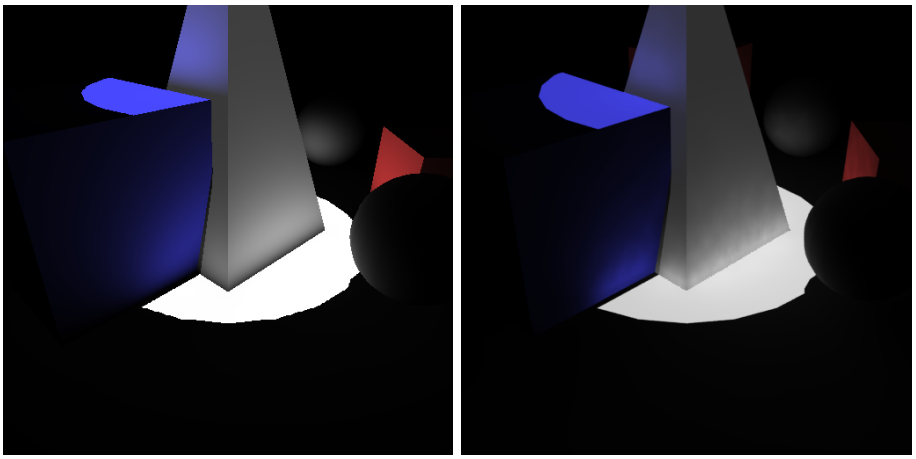


Figure 2.2: Lambert cosine diffuse reflection.

In figure 2.3 the second scene is shown. The scene can also be seen in 1.1. The only directly illuminated part of the scene is inside the bright circle. The

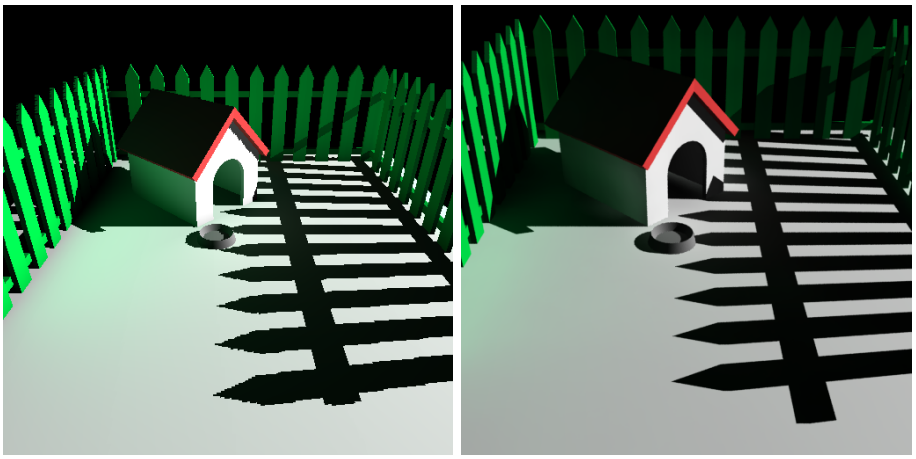


(a) Image based global illumination result. (b) Mental Ray Photon mapping result.

Figure 2.3: Rendering of the second scene with $nc = \frac{|P_c|}{150}$

rest of the scene is only light by indirect illumination. Once again the ambient occlusion can be seen with this approach. The photon mapping additionally suffers from noisy lighting.

Except for the slightly different camera angle in figure 2.4, the Photon mapping approach shows some light bleeding artifacts through the interior wall of the dog-house. On the other hand, the photon-tracing approach does not seem to suffer from green light-bleeding artifacts inside the doghouse, which is clearly visible on the interior wall of the dog-house in 2.5a.



(a) Image based global illumination result. (b) Mental Ray Photon mapping result.

Figure 2.4: Rendering of the third scene with $nc = \frac{|P_c|}{150}$

2.3.1 Performance

Four different methods of increasing the algorithms performance were implemented:

- Naive approach
- Optimized naive approach which pre-calculates values that are used multiple times, such as conversion of the depth values stored inside the image buffers to position values.
- Intensity discard approach using a quad-tree on the image planes.
- Distance discard approach using a distance quad-tree on the image planes.

The table 2.1 shows some run-times in seconds on a Intel Core i7 2.4 GHz without using multi-threading for the algorithm. For larger image resolutions the runtime for the second scene was 1284,193 seconds for a resolution of 256 by 256 and 19947,639 seconds for 512 by 512 pixels.

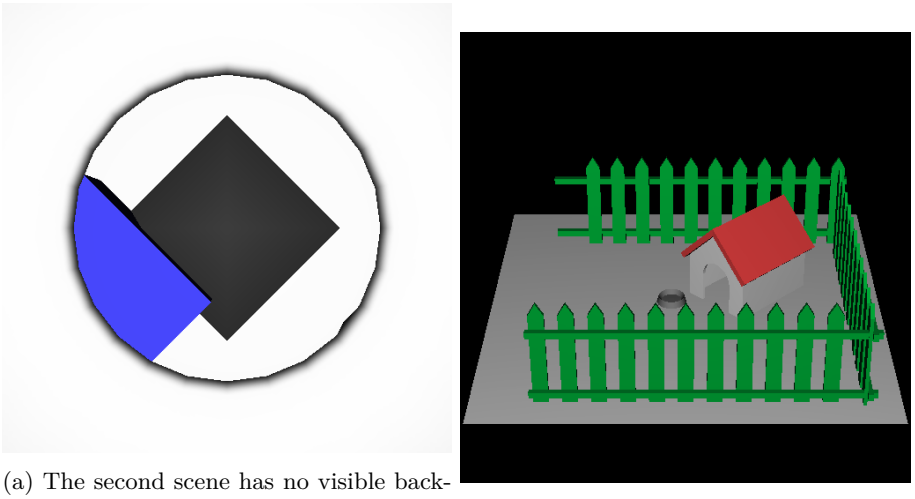
The second scene is ill suited for the proposed performance optimization methods. This is the case, because, the second scene was designed to have large regions of indirect illumination, while the regions with direct illumination were kept relatively small. Therefore, the overhead of generating and traversing a quad-tree does not pay off. The intensity discard method relies on black background regions in the light-source perspective, these regions are also smaller in the second scene.

The distance discard approach can additionally be parametrized with a scaling factor to the light fall-off distance. In theory a value of 256 considers all possible 8-bit values. A value of 128 was chosen, since the result suffices. A further decrease of this scaling factor increases the runtime of the approach, while losing accuracy.

	naive	optimized	intensity discard	distance discard
First scene (64x64)	19,254	12,763	8,944	7,283
Second scene (64x64)	4,098	2,657	3,912	4,252
Third scene (64x64)	4,054	2,660	2,068	1,822
First scene (128x128)	330,455	203,659	146,917	130,384
Second scene (128x128)	60,647	36,553	60,206	71,941
Third scene (128x128)	61,532	37,051	29,976	24,826

Table 2.1: Runtime in seconds for the naive, optimized and intensity and distance discard methods. The first scene has 6 light-sources, while the other scenes are light by one light-source.

Figure 2.5 shows that in the second scene, there are no background regions visible from the light perspective. Therefore the distance discard approach can not optimize the program execution.



(a) The second scene has no visible background regions visible from lights perspective.

(b) The third scene has larger background regions visible from the lights perspective.

Figure 2.5: Light source perspectives for the second and thrird scene

2.4 Conclusion

A comparison to the mental ray Photon mapping reference was shown in figure 2.1, figure 2.3 and figure 2.4.

The reference depends on the number of photons used. The larger the number of photons, the longer the runtime. In addition there is a quality parameter for smoothing.

The Photon mapping is stochastic in the positioning of the Photons. Therefore, the noise is random. On the other hand, the implemented approach introduces aliasing noise, similar to shadow mapping.

As seen on figure 2.4, the Photon map implementation suffers from light leaking artifacts. These can be fixed, but in contrary to the Photon mapping, the implemented approach does not suffer from light leaking.

The implemented approach does not consider shadow mapping for the second bounce.

Compared to photon-mapping, the implemented approach would require less effort for non-perfectly-diffuse surfaces. The photon mapping approach would require to store the light directions for every photon. Due to its stochastic approach

2.5 Future work

The advantage of the approach is, that for two bounces, the lighting calculation is theoretically correct, except for an aliasing error through rasterization. All

the required information is available in the final image and the images as seen from the light-sources.

The drawbacks of the approach are:

- Anti-aliasing needs to be implemented.
- The algorithm is relatively slow on the CPU.
- The shadow mapping bias depends on the resolution of the rendered image.
- A more rigorous handling of the size of pixels may be necessary. Currently, the algorithm simply uses a constant normalization factor for all pixels.

A GPU implementation would increase the performance. Mip-mapping with an additional variance could be used to simplify the image planes and therefore significantly reduce the effort required during the summation of the pixels.

The algorithm is easily extendable to support non-perfectly-diffuse surfaces.

For the perfectly diffuse case, the algorithm could be used with an illuminating environment map, for example a real photo.

It would be possible to use the algorithm for an offline pre-calculation of global illumination. Instead of rendering the algorithm for an image plane, it would be possible to render the texture space of an object, to generate a light-map for the object.

Currently the quad-trees were used simply as a discard mechanism. If a region of the image does not influence the result, the quad-tree will skip these parts earlier. In addition it would be possible to generate a quad-tree representation of the positions, normals and intensities. The internal nodes could then be used to approximate low intensity regions, to further speed up the algorithm while reducing accuracy.

2.6 Appendix: Code FAQ

The window and resolution can be modified in: WindowWrapper.java.

public static final float WINDOWSCALE = 8.0f; // additional upscaling public static final int WINDOWX = 16; // resolution x direction public static final int WINDOWY = 16; // resolution y direction

[0]Source: `http://en.wikipedia.org/wiki/Diffuse_reflection`

Bibliography

[FG03] Kaspar Fischer and Bernd Gartner. The smallest enclosing ball of balls: Combinatorial structure and algorithms. In *Proceedings of the Nineteenth Annual Symposium on Computational Geometry*, SCG '03, pages 292–301, New York, NY, USA, 2003. ACM.

[Wel91] Emo Welzl. Smallest enclosing disks (balls and ellipsoids). In *Results and New Trends in Computer Science*, pages 359–370. Springer-Verlag, 1991.