Bounded Radiosity with Affine Arithmetic

Studienarbeit im Fach Informatik

vorgelegt von

Hendrik Lensch

geb. am 30. Mai 1973 in Bremervörde

angefertigt am

Institut für Mathematische Maschinen und Datenverarbeitung
Lehrstuhl für Graphische Datenverarbeitung
Friedrich-Alexander-Universität Erlangen-Nürnberg

Betreuer: Wolfgang Heidrich und Marc Stamminger

Beginn der Arbeit: 1. Januar 1998

Abgabe der Arbeit: 23. September 1998

Contents

Chapter 1

Introduction

Through the last two decades, one goal of computer graphics research has been, and still is, to compute photorealistic images from descriptions of three-dimensional environments (e.g. architectural models). This way one can use the pictures for Virtual Reality like in computer games and simulators or examine visual effects in the environment without explicitly building up a physical model.

To gain an image, the sources of light in a scene, the light transport and the light's interaction with surfaces have to be considered. Light emitted from one surface may be reflected arbitrary times illuminating surfaces that are not directly visible to the emitting surfaces. This is usually called *global illumination*. The emission and reflection of light on surfaces and the resulting global illumination can be described by the *rendering equation* [12]. Due to its complexity a solution of this integral equation can be approximated only with high computational cost. Simplifications are possible by restricting the kind of reflection in the environmental model.

Radiosity[1] methods reduce the complexity by solving only for interreflection of light between ideal diffuse surfaces, which means incident light is equally distributed in all outgoing directions. The first algorithm of this kind has been presented by Goral *et al.* [7] in 1984. These methods are finite-element methods that solve the corresponding *radiosity equation*. The finite elements, patches, are subdivided parts of the surfaces in the scene which emit and reflect light. In a first step the light sources emit their light and illuminate some patches which partly reflect it and act as light sources in the next step, and so on. So, the light is spread through the scene until equilibrium is reached.

As light may be transported between two arbitrary patches the number of transports is quadratic to the number of given patches. To overcome this problem, in Hierarchical Radiosity [8] first an approximation of the transport between two surfaces of the scene is computed; only if the quality of the approximation is below some user defined threshold the surfaces will be subdivided and a new approximation for the transport between the resulting smaller patches

[1]radiosity: a measure of the radiant energy leaving a surface per unit area per unit time

is computed. Otherwise the transport will take place at the actual level of subdivision. By this, the patches are adaptively subdivided and the cost is reduced to be $O(n)$ in the number of patches. One step further, it is possible to start the subdivision not at the level of the given surfaces but on groups of them, called clusters. Of course, it is more involving to approximate the interaction between clusters as they tend to be more inhomogeneous than single surfaces, but it is less expensive to compute one interaction at this level than several at the lower. There exists a variety of algorithms which use radiosity to solve the global illumination problem.

One way to approximate the transport of light from one patch to another is to calculate conservative upper and lower bounds for the transport, yielding upper and lower bounds for the illumination of each patch [13]. The quality of the approximation now is correlated to the error of the interval. These intervals can be handled with standard *interval arithmetic* (IA). But it is not convenient to use IA the whole way because IA is too conservative in long computational chains resulting in unnecessary wide intervals. Stamminger *et al.* [20] suggested one approach to determine an error interval for each involved quantity, sometimes calculating the upper bound in another way than the lower one to obtain tighter bounds. Chapter 2 reflects the radiosity method as it is necessary for this thesis.

Often, intervals are used to represent one quantity whose exact value is not known due to some source of uncertainty or error. Combining two different intervals by an arithmetic operation the resulting interval normally depends on both sources of error. This is not taken into account with standard IA. In 1993 Comba and Stolfi [3] proposed the use of *affine arithmetic* (AA) to address this problem by considering correlations between quantities. Of course, AA is more expensive than IA but the extra cost may be justified by gaining of higher accuracy. AA will be further described in Chapter 3.

This thesis investigates, describes and implements the application of AA to one radiosity algorithm. AA is used to represent the geometry of the surfaces, to perform visibility tests, to calculate the bounds for the transport and to represent the illumination of each patch, and so on. For some quantities it is not possible to compute its interval with arithmetic operations; so, AA cannot be applied here and the interval must be assembled by finding an upper bound and a lower bound separately introducing one totally new source of uncertainty which is not correlated to any yet existing. Compared to Stamminger's implementation of bounded radiosity, using AA the bounds on the transported energy are noticeable tightened and thus less interactions and subdivisions are necessary in order to fulfill the same quality criterion. Even if the number of simulated interactions is nearly halved in some cases the higher complexity of affine arithmetic operations causes the entire solution time to become larger.

Chapter 4 explains in detail how to perform the radiosity approach with AA and how to control the quality of the computation. The implementation within the Vision system [17], a system for global illumination computations, is described in Chapter 5. In Chapter 6 further results are summarized.

Chapter 2

Radiosity

In 1984 the first algorithm using radiosity to solve the global illumination problem was introduced by Goral *et al.* [7]. All influence of participating media, interference and refraction has been ignored. Since then there has been much effort to augment and to speed up this method (see [2], [16]). *Radiosity* is actually a measure of radiant energy.

2.1 Radiance, Irradiance and Radiosity

Radiance, the angular flux density, denoted by L, is the most important quantity to describe radiant energy transfer. It is defined as the power at some point traveling in a given direction, per unit projected area in this direction, per unit time, per unit solid angle. Its unit is $[Wm^{-2}sr^{-1}]$.

As one might expect, within radiosity-methods *radiosity*[1] is an important quantity. It is defined as the total radiant energy leaving a point on a surface, per unit area, whereas *irradiance* is the total radiant energy falling on a surface point, per unit area. The unit of these two quantities is $[Wm^{-2}]$. Usually, radiosity is denoted by B and irradiance by E. They can be expressed in terms of radiance as follows:

$$B(\underline{x}) = \int_\Omega L_o(\underline{x}, \underline{\omega}) \cos\theta d\underline{\omega}, \tag{2.1}$$

respectively

$$E(\underline{x}) = \int_\Omega L_i(\underline{x}, \underline{\omega}) \cos\theta d\underline{\omega}, \tag{2.2}$$

integrating radiance over Ω, the set of directions in the hemisphere covering the surface at point $\underline{x}$. θ denotes the angle between the direction and the surface's normal and L_o refers to the outgoing and L_i to the incident radiance. Radiosity and Irradiance are functions of the position on the surfaces and of wavelength only. Using radiosity to describe the illumination of a surface at one point, only ideal diffuse emitting or reflecting (Lambertian) surfaces can be considered because a directional dependency is needed, like it is included in radiance, to represent illumination produced by specular reflection.

[1] Another term is *radiant exitance*

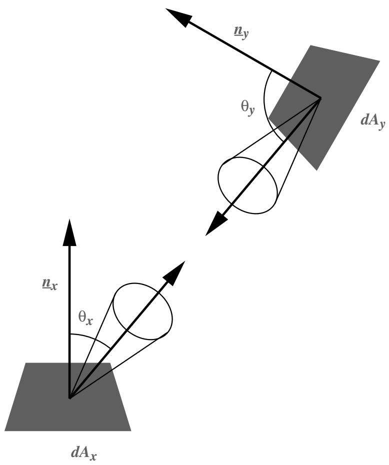


Figure 2.1: Geometrical properties of the energy transport

2.2 The Radiosity Equation

Kajiya presented in [12] the *rendering equation* which expresses the energy equilibrium in a scene of arbitrary opaque surfaces:

$$L(\underline{y},\underline{\omega}) = L_e(\underline{y},\underline{\omega}) + \int_S \rho(\underline{\omega}_{xy},\underline{y},\underline{\omega})G(\underline{x},\underline{y})L(\underline{x},\underline{\omega}_{xy})dA_x \tag{2.3}$$

It is an integral equation over the surfaces S, with the following definitions for the other quantities: $\underline{\omega}_{xy}$ is the direction from $\underline{x}$ to $\underline{y}$; $L_e(\underline{y},\underline{\omega})$ is the radiance emitted from the surface at point $\underline{y}$ in direction $\underline{\omega}$; $\rho(\underline{\omega}_{xy},\underline{y},\underline{\omega})$ stands for the reflection properties of the surface (BRDF) at point $\underline{y}$, that is the fraction of radiance incident from direction $\underline{\omega}_{xy}$ that will be reflected in the outgoing direction $\underline{\omega}$; $G(\underline{x},\underline{y})$ refers to the geometrical properties of the energy transport between $\underline{x}$ and $\underline{y}$, see Figure 2.1. It also includes the visibility $V(\underline{x},\underline{y})$ whose range is between 0 ($\underline{y}$ is not at all visible from $\underline{x}$) and 1 (there is no occluder):

$$G(\underline{x},\underline{y}) = \frac{\cos\theta_x \cos\theta_y}{\|\underline{x}-\underline{y}\|^2} V(\underline{x},\underline{y}) \tag{2.4}$$

Most illumination computations may be seen as a more or less restricted approximation of a solution of the rendering equation. For ideal diffuse surfaces the quantities $L(\underline{y},\underline{\omega})$, $L_e(\underline{y},\underline{\omega})$ and $\rho(\underline{\omega}_{xy},\underline{y},\underline{\omega})$ are constant for all directions at some point. So, global illumination in an environment consisting only of Lambertian surfaces can be computed by solving the *radiosity equation* [7]:

$$B(\underline{y}) \quad = \quad E(\underline{y}) + \rho(\underline{y}) \int_S \frac{1}{\pi} G(\underline{x},\underline{y}) B(\underline{x}) dA_x \tag{2.5}$$

2.3 A Discrete Formulation

One way to solve integral equations are finite-element methods. The domain of the integral is subdivided into finite elements and the solution is approximated by projecting the exact solution onto a finite dimensional function space across each element. In classical radiosity algorithms, like in this thesis, most often the radiosity, $B(\underline{x})$, the emitted irradiance $E(\underline{x})$ and the diffuse reflectance $\rho(\underline{x})$ are considered to be piecewise constant on each element, but more elaborated function spaces are possible (see [26] or [24]). Suppose, the scene's surfaces are subdivided into N elements, patches, $(P_i)_{i=1\ldots N}$; by this the radiosity equation is discretized:

$$B_i = E_i + \rho_i \sum_{j=1}^{N} F_{ij} B_j, \tag{2.6}$$

where F_{ij} refers to the form factor that is described in the next section.

2.4 The Form Factor

The form factor F_{ij} is the proportion of the total power leaving Patch P_i that reaches patch P_j. As may be seen from Equation 2.5 the *differential form factor* between two infinitesimal small elements for the unoccluded case is given by

$$dF_{di \to dj} = \frac{\cos\theta_i \cos\theta_j}{\pi r^2} dA_j, \tag{2.7}$$

with r denoting the distance between the elements. To get a form factor for two patches, or element-to-element form factor, one needs to integrate over the sending patch and to average over the receiving, yielding a double area integral:

$$\begin{aligned} F_{ij} &= \frac{1}{\mid A_j \mid} \int_{\underline{x} \in P_i} \int_{\underline{y} \in P_j} \frac{\cos\theta_x \cos\theta_y}{\pi r^2} V(\underline{x}, \underline{y}) d\underline{y} d\underline{x} \quad \text{or} \\ &= \frac{1}{\mid A_j \mid} \int_{A_i} \int_{A_j} \frac{\cos\theta_i \cos\theta_j}{\pi r^2} V_{ij} dA_j dA_i \end{aligned} \tag{2.8}$$

Computing the form factor is one of the most challenging tasks in radiosity algorithms because analytic closed form solutions only exist for a few simple unoccluded cases, like polygon-to-polygon or point-to-disk form factors. But other methods are known to approximate the form factor. Of course it can be sampled using ray casting with all its problems. In Section 2.7 upper and lower bounds on the form factor are derived from the geometrical description of the surfaces. Projection methods apply *Nusselt's analogy* to determine the point-to-patch form factor (e.g. the hemi-cube algorithm):

> F_{xP_j}, the form factor from the point x to the element P_j, is proportional to the fraction of the area of the unit disc that is covered by the double projection of the patch P_j first onto the hemisphere centered at x and then orthogonally down onto its base plane.

More detailed information on properties of the form factor and methods to compute it can be found in [2] or [16].

2.5 Solving the System of Equations

Now we have established a system of equations; one equation for the radiosity of each patch. Those systems may be solved by different methods but because of the great amount of equations iterative methods are best suited. For the *Gauss-Seidel algorithm* and the *Southwell relaxation* there even exist physical interpretations of each step in the iteration.

In the Gauss-Seidel algorithm in each iteration step one patch is chosen for which the radiosity is calculated using Equation 2.6, that means this patch collects energy from all other patches, called *gathering*. In the next step another patch is chosen, now using the newly calculated values. This process is iterated until the radiosity values converge.

The Southwell iteration is named *shooting* within the radiosity context. In each step one patch is chosen that distributes all its unshot radiosity to all other patches, adding some amount to their unshot radiosity. If always that path with the most unshot radiosity is taken this algorithm yields rough approximations faster.

After convergence or with some more effort even during the iteration the results can be displayed using standard graphics hardware. Here the absence of directional dependency is an advantage of the radiosity method because views of the scene can be produced from arbitrary positions without recalculating the radiosity values.

2.6 Hierarchical Radiosity

As is quite clear, the quality of the subdivision of the given surfaces into patches restricts the accuracy of the solution. Discontinuities or rapid changes in the radiosity across one surface should be considered by an appropriate subdivision. One approach is to generate patches whose borders are aligned with the discontinuity in a preprocessing step (*discontinuity meshing* [9]). Another is to subdivide the patches adaptively where a rapid change of the radiosity function has been detected during the simulation process.

No matter which way is chosen, using Equation 2.6 as it is suffers from one problem: the full matrix of N^2 form factors F_{ij} has to be computed if N is the number of given elements used in the computation, yielding a complexity of $O(N^2)$. This limits the accuracy of the results due to computational cost and storage.

In 1989 Hanrahan *et al.* presented an algorithm for *hierarchical radiosity* that reduces the effort to $O(N)$ where N refers to the number of resulting patches after the subdivision [8]. The surfaces are adaptively subdivided into patches that may again be subdivided introducing a new level of subdivision. By this a hierarchical structure is constructed. The main idea of the hierarchical radiosity algorithm is that the levels of subdivision for the sending and for the receiving surface are determined separately for each transport of radiosity by a quality criterion. Even transport between two different levels within the hierarchy is possible. First, each pair of initial surfaces at the coarsest level is connected by a link. Only if the error of the energy exchange introduced by computing the form factor for this link is above a certain threshold further subdivision is necessary. Thus, a couple of interactions at a lower level can be avoided by a single one at the next coarser level.

Because each surface may gather radiosity at arbitrary levels within its own subdivision tree during one iteration step and every patch within this structure possibly acts as a sender in the next iteration the whole gathered radiosity must be collected from and propagated to all subpatches. In the first step (*push* operation) the gathered radiosity of a higher level is propagated down the tree where each child adds its own radiosity before passing it to the next lower level. The second step (*pull* operation) updates bottom up the radiosity of each parent patch by computing the area-weighted sum of the radiosity of its children.

The number of resulting links depends extremely on the decision on the subdivision of a surface. The part of the algorithm that performs this decision is commonly called *oracle*. Often, *BF refinement* is applied where subdivision is proposed if the product of the radiosity of the sender B and the form factor F lies above some user defined value. For bright patches the form factor is forced to become small and in that way more precise, whereas for patches with lower radiosity the form factor will be computed less exactly.

For complex scenes the initial linking of the given surfaces becomes a restricting factor. At the coarsest level in the hierarchy there are still s^2 links. The number of initial links can only be reduced if we do not link the surfaces themselves but groups of them, *clusters*, or groups of groups, yielding new levels [18]. Thus, the hierarchy is extended upwards with only one node at the top, the collection of all objects present in the scene. Two problems must be solved before using clusters in a radiosity algorithm: First, those surfaces must be selected that form one cluster; this may be done automatically using spatial information or by hand. Second, more work has to be done to approximate the energy transport correctly (see [18] or [15]) since a cluster may not be treated like a Lambertian diffuse surface. Due to the different orientation of the contained patches incident radiosity will not be reflected equally in all directions. In addition, total visibility between some object and some cluster does not mean total visibility between this object and all the surfaces in the cluster.

2.7 Bounded Radiosity

Because there is no analytical solution for the form factor for arbitrary shaped surfaces, one common approach is to compute it by a few point samples. The accuracy will improve as one increases the number of sample points but still the results depend on the chosen samples.

As mentioned earlier instead of computing one more or less exact value for the form factor it can as well be determined by finding conservative[2] upper and lower bounds for it. Lischinski

[2]This means the exact value of the form factor is guaranteed to lie in the interval between upper and lower

et al. [13] proposed how to use upper and lower bounds on the form factor to derive bounds on the radiosity solution in a hierarchical radiosity algorithm. Because those bounds can be found even for curved surfaces no initial triangulation into planar patches is needed and arbitrary surfaces can be handled directly.

Let $F_{ij\,\min}$ denote the infimum on the point-to-area form factor and let $K_{ij\,\min} = \rho_{i\,\min} F_{ij\,\min}$. For the lower bounds on the radiosity $B_1, \ldots, B_n$ a solution is found easily by solving the following linear system of n equations:

$$(I - K_{\min}) B_{\min} = E_{\min} \tag{2.9}$$

Determining the uppers bounds the same way runs into difficulty. Let $K_{ij\,\max} = \rho_{i\,\max} F_{ij\,\max}$. The sum of the entries of one row in the matrix $K_{\max}$ can now exceed 1 because the entries are upper bounds on the form factor. Thus, the matrix $(I - K_{\max})$ is not necessarily diagonally dominant and can even be singular; iterative solution methods like Gauss-Seidel are no longer guaranteed to converge for the equation

$$(I - K_{\max}) B_{\max} = E_{\max}. \tag{2.10}$$

Lischinski *et al.* address this situation by neglecting the smallest form factors in each row until the sum no longer exceeds 1. This of course requires some sorting and thus all form factors for one patch have to be accessible at the same time.

Bounds on the form factor also enables an estimation of the error of the transferred energy by a link that can be applied as a subdivision criterion in a hierarchical radiosity algorithm. In [20] and [21] Stamminger *et al.* not only took up this idea but also proposed concrete ways to obtain those bounds for each form factor. Surfaces of arbitrary shape and clusters can be handled as long as they support information about their spatial extent and their normals. In their paper the bundle of normals of one object was collected by a cone of normals that also bounds the location of the surface, see [20] for more details.

For each term within the form factor kernel between sender i and receiver j

$$k_{ij}(\underline{x}, \underline{y}) = \frac{\cos\theta_x \cos\theta_y}{\pi r^2} V(\underline{x}, \underline{y}) \tag{2.11}$$

upper and lower bounds are computed separately yielding $(\cos\theta_i)_{\min}$, $(\cos\theta_i)_{\max}$, $(\cos\theta_j)_{\min}$, $(\cos\theta_j)_{\max}$, $r_{\min}$, $r_{\max}$, $V_{\min}$ and $V_{\max}$. By this, bounds on the form factor kernel can be found easily using standard interval arithmetic:

$$\begin{aligned}
k_{ij\,\min} &\leq k_{ij}(\underline{x}, \underline{y}) \leq k_{ij\,\max}, \text{ with} \\
k_{ij\,\min} &= \frac{(\cos\theta_i)_{\min} (\cos\theta_j)_{\min}}{\pi r_{\max}^2} V_{\min} \text{ and} \\
k_{ij\,\max} &= \frac{(\cos\theta_i)_{\max} (\cos\theta_j)_{\max}}{\pi r_{\min}^2} V_{\max}
\end{aligned}$$

bound.

But this is just the kernel; to result in the final patch to patch form factor the kernel has to be averaged over the receiver and integrated over the domain of the sender. Because the average value is guaranteed to lie within upper and lower bound only integration is left which is done by multiplication with the sender's area A_i. Thus, bounds on the reveiver's irradiance B_j induced by the sender's radiosity B_i are given by

$$\begin{aligned} B_{\min} &\leq B_j(\underline{x}, \underline{y}) \leq B_{\max}, \text{ with} \\ B_{\min} &= k_{ij\,\min} A_i B_{i\,\min} \quad \text{and} \\ B_{\max} &= k_{ij\,\max} A_i B_{i\,\max}. \end{aligned}$$

Now, the error of transported radiosity by a link can be computed. Suppose that the approximate value B_j is the center of the interval $[B_{\min}, B_{\max}]$, the errors in the L_1, L_2, and L_∞ norms are given by

$$\begin{aligned} \|B_j(\underline{x}, \underline{y}) - B_j\|_1 &= \int_D \mid B_j(\underline{x}, \underline{y}) - B_j \mid dx \leq A_i(B_j(\underline{x}, \underline{y}) - B_j)/2 \\ \|B_j(\underline{x}, \underline{y}) - B_j\|_2 &= \int_D \mid B_j(\underline{x}, \underline{y}) - B_j \mid^2 dx \leq A_i(B_j(\underline{x}, \underline{y}) - B_j)^2/4 \\ \|B_j(\underline{x}, \underline{y}) - B_j\|_\infty &= \max_D \mid B_j(\underline{x}, \underline{y}) - B_j \mid \leq (B_j(\underline{x}, \underline{y}) - B_j)/2 \end{aligned}$$

The concrete way of how to determine bounds for the terms of Equation 2.11 is explained in Chapter 4 since this thesis uses these ideas of Stamminger *et al.* but computes the bounds with affine arithmetic instead of interval arithmetic.

2.8 Conclusion

Bounded radiosity approximates a solution of the radiosity Equation for arbitrary surfaces by applying the hierarchical radiosity method with clusters. The error of the transported radiosity by a link decides on subdivision. This error is determined by finding upper and lower bounds on the form factor computed from bounds on the terms of Equations 2.11 with standard interval arithmetic. IA allows separate, and thus quite precise, computation of the upper and the lower bound for each of these terms but combining those bounds by interval arithmetic ignores any correlation between them which leads to unnecessary wide bounds on the form factor and that way unnecessary subdivision. Because affine arithmetic by contrast during computation keeps track of the correlation between two intervals that share some source of uncertainty, one can expect tighter bounds using AA instead of IA as is done and verified in this thesis.

Chapter 3

Affine Arithmetic

Assume that there is a quantity of which the exact value is unknown but an interval is given that is guaranteed to contain the unknown value. Applying an arithmetic operation to such an interval the result of course should be computed in such a way that if the same operation is applied to the unknown value its result is still guaranteed to lie within the computed interval.

3.1 Interval Arithmetic

Standard interval arithmetic (IA), also known as interval analysis introduced by R. E. Moore [14], performs those arithmetic operations by inspecting the upper and the lower bound of the initial interval and constructing the result by efficiently computing new upper and lower bounds. This way for one operation applied to only one interval (e.g. dividing a scalar by an interval or computing the cosine, ...) the resulting interval can be computed as tight as possible. However, standard IA suffers from the problem that the computed intervals tend to be much wider than the true range of the corresponding quantities, particular in long computational chains.

Intervals may be represented in two ways, by their minimum and maximum value $\overline{x} = [x_{\min}, x_{\max}]$ or by its center value and some error $\overline{x} = x_{center} \pm x_{err}$. Given the intervals $\overline{x} = [x_{\min}, x_{\max}]$ and $\overline{y} = [y_{\min}, y_{\max}]$, the sum and the difference of two intervals may be defined as

$$\begin{aligned}\overline{x} + \overline{y} &= [x_{\min} + y_{\min}, x_{\max} + y_{\max}] = x_{center} + y_{center} \pm x_{err} \pm y_{err} \\ \overline{x} - \overline{y} &= [x_{\min} - y_{\max}, x_{\max} - y_{\min}] = x_{center} - y_{center} \pm x_{err} \pm y_{err}\end{aligned}$$

Other operations can be defined as simple as this and they may be combined to form more complex functions.

One extreme example for unnecessary wide bounds after some computation is the difference of an interval and itself:

$$\overline{x} - \overline{x} = [x_{\min} - x_{\max}, x_{\max} - x_{\min}] = \pm 2x_{err} \tag{3.1}$$

which is twice as wide as $\overline{x}$ and never equals $[0,0]$ because the representation does not contain any information about the fact that $\overline{x}$ and $\overline{x}$ possess the identical error. As one may imagine, the deviation of the computed range of an interval and the true range of the corresponding quantity get worse with the number of operations.

3.2 Definition of an Affine Form

To address this problem Comba and Stolfi proposed a new method called affine arithmetic (AA) [3]. In addition to conservative bounds AA keeps track of the correlation between parameters to functions and their results. That way, errors introduced by some operation may cancel out in further computational steps.

Introduced errors or some other source of uncertainty causes that the exact value of some quantity cannot be determined. If two of those uncertain quantities are combined e.g. by addition the result will be imprecise, depending on both sources of uncertainty. This leads to the definition of an *affine form*, a first-degree polynomial

$$\widehat{x} = x_0 + x_1\varepsilon_1 + x_2\varepsilon_2 + \cdots + x_n\varepsilon_n \quad \text{, where} \tag{3.2}$$

each ε_i, called error variable, represents one independent source of uncertainty. Its value is unknown but bounded by the interval $U = [-1, +1]$. In addition, the x_i are known real coefficients. Each x_i may be seen as the magnitude of dependency of the affine form $\widehat{x}$ on the error i. Two different affine forms $\widehat{x}$ and $\widehat{y}$ may share several of their error variables expressing some correlation between their underlying quantities x and y. x_0 denotes the central value of the affine form $\widehat{x}$. Approximating $\widehat{x}$ by x_0 introduces the smallest possible error.

3.3 Conversion between IA and AA

Assume that the interval $\overline{x} = [x_{\min}, x_{\max}]$ represents some unknown quantity x; an affine form for it can be generated as

$$\begin{aligned} \widehat{x} &= x_0 + x_k\varepsilon_k \quad \text{, with} \\ x_0 &= \frac{x_{\max} + x_{\min}}{2} \quad \text{and} \quad x_k = \frac{x_{\max} - x_{\min}}{2}. \end{aligned} \tag{3.3}$$

If no further information is given and since $\overline{x}$ does not constrain the value of x with respect to those of other variables, a brand new error variable ε_k has to be introduced that has not yet been used by any other quantity.

On the other side the range and thus the corresponding interval of an affine form is given by

$$[\widehat{x}] = [x_0 - \xi, x_0 + \xi], \quad \xi = \sum_{i=1}^{n} | x_i | \tag{3.4}$$

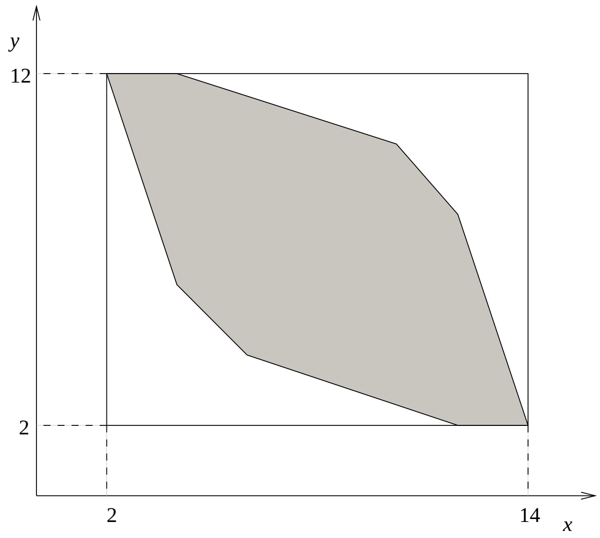


Figure 3.1: possible range of (x, y) using AA and IA

which is the smallest interval that contains all possible values of x obtained if all ε_i are chosen independently out of the interval $[-1, +1]$.

The advantage of the representation with affine forms over standard intervals may be seen in Figure 3.1. The pair $(\widehat{x}, \widehat{y})$ of the affine forms

$$\begin{aligned} \widehat{x} &= 8 + 1\varepsilon_1 + 2\varepsilon_2 + 1\varepsilon_3 - 1\varepsilon_4 \\ \widehat{y} &= 7 - 1\varepsilon_2 - 1\varepsilon_3 + 3\varepsilon_4 \end{aligned} \tag{3.5}$$

is displayed. The additional information about the correlation between $\widehat{x}$ and $\widehat{y}$ restricts the possible real values of (x, y) to lie in the region of $\mathbf{R}^2$ which is colored grey in the Figure. It is considerable smaller than the corresponding region generated with IA, that is $[2, 14] \times [2, 12]$.

3.4 Computing Formulas with Affine Forms

Formulas on real numbers may be applied to affine forms by replacing every elementary operation $z \leftarrow f(x, y)$ on real numbers by a corresponding operation on affine forms $\widehat{z} \leftarrow \widehat{f}(\widehat{x}, \widehat{y})$. Operations on affine forms can be divided into two groups. The first group are affine operations which can be calculated without introducing new errors to the result. But more difficult are non-affine operations that only can be approximated by an affine expression in such a way that some error must be added to guarantee conservative bounds.

With affine arithmetic in addition an automatic treatment of round-off errors is possible but this is neglected in this thesis because the errors introduced by other circumstances tend to be more important in the radiosity context.

3.4.1 Affine Operations

Affine operations act on affine forms the same way as on polynomials. The result is extended to contain all terms and similar terms are collected. Thus, given two affine forms $\hat{x} = x_0 + x_1\varepsilon_1 + x_2\varepsilon_2 + \cdots + x_n\varepsilon_n$ and $\hat{y} = y_0 + y_1\varepsilon_1 + y_2\varepsilon_2 + \cdots + y_n\varepsilon_n$, and some scalar α, the affine operations are

$$\begin{aligned} \hat{x} \pm \hat{y} &= (x_0 \pm y_0) + (x_1 \pm y_1)\varepsilon_1 + \cdots + (x_n \pm y_n)\varepsilon_n & (3.6)\\ \alpha\hat{x} &= (\alpha x_0) + (\alpha x_1)\varepsilon_1 + (\alpha x_2)\varepsilon_2 + \cdots + (\alpha x_n)\varepsilon_n & (3.7)\\ \alpha \pm \hat{x} &= (\alpha \pm x_0) + x_1\varepsilon_1 + x_2\varepsilon_2 + \cdots + x_n\varepsilon_n & (3.8) \end{aligned}$$

Note that no information about the result that can be derived from $\hat{x}$ and $\hat{y}$ is lost. This way formulas consisting only of affine operations can be performed in AA yielding exact results. Also $\hat{x} - \hat{x}$ is identically zero.

3.4.2 Non-affine Operations

Since non-affine operations on first-degree polynomials normally do not result in first-degree polynomials, a resulting affine form can only represent an approximation. For each non-affine operation $z = f^*(\hat{x}, \hat{y})$ the corresponding formulation in AA looks like

$$\hat{z} = \hat{f^a}(\hat{x}, \hat{y}) + z_k\varepsilon_k \tag{3.9}$$

introducing a new error variable ε_k. $\hat{f^a}(\hat{x}, \hat{y})$ is a first-degree polynomial, an affine form, approximating $f^*(\hat{x}, \hat{y})$ reasonable well over the domain ($[\hat{x}]$, $[\hat{y}]$). The component z_k denotes the upper bound on the difference between f^a and f^* over the domain.

$$z_k = \max\{\mid f^*(x, y) - f^a(x, y) \mid : x \in [\hat{x}], y \in [\hat{y}]\} \tag{3.10}$$

There exists no constraint any more between ε_k and the initial error variables $\varepsilon_0, \ldots, \varepsilon_n$. Thus, further operations on $\hat{z}$ will yield less precise results than necessary due to this loss of information.

To minimize this influence Comba and Stolfi proposed that f^a should approximate f^* best in the Chebyshev sense of minimizing the maximum error z_k. Note that only the newly introduced error is minimized this way, not the range of the resulting interval. Sometimes, a better choice is to keep the range within defined bounds even if the error gets larger. For example, the square of an interval should of course be positive if it is used as a denominator later.

3.4.3 Multiplication

Because multiplication is applied quite often within this thesis, e.g. to scale vectors of affine forms, and to get a feeling for the upcoming error, the way of how to perform it is given here.

Like multiplication of polynomials the product of two affine forms $\widehat{x}$ and $\widehat{y}$ can be written as

$$\begin{aligned} f(\widehat{x},\widehat{y}) &= \widehat{x}\cdot\widehat{y} \\ &= (x_0+\sum_{i=1}^{n} x_i\varepsilon_i)\cdot(y_0+\sum_{i=1}^{n} y_i\varepsilon_i) \\ &= x_0y_0+\sum_{i=1}^{n}(x_0y_i+y_0x_i)\varepsilon_i \\ &\quad +(\sum_{i=1}^{n} x_i\varepsilon_i)\cdot(\sum_{i=1}^{n} y_i\varepsilon_i) \end{aligned} \tag{3.11}$$

This function is best approximated if we choose

$$f^a(\widehat{x},\widehat{y}) = x_0y_0+\sum_{i=1}^{n}(x_0y_i+y_0x_i)\varepsilon_i \tag{3.12}$$

The remaining quadratic term $(\sum_{i=1}^{n} x_i\varepsilon_i)\cdot(\sum_{i=1}^{n} y_i\varepsilon_i)$ can be bounded[1] by the interval $[-uv,+uv]$, with

$$\begin{aligned} u &= \sum_{i=1}^{n} \mid x_i \mid = \frac{\widehat{x}_{\max}-\widehat{x}_{\min}}{2} \\ v &= \sum_{i=1}^{n} \mid y_i \mid = \frac{\widehat{y}_{\max}-\widehat{y}_{\min}}{2} \end{aligned} \tag{3.13}$$

Thus, multiplication of two affine forms is given by

$$\begin{aligned} \widehat{z} &= \widehat{x}\cdot\widehat{y} \\ &= x_0y_0+\sum_{i=1}^{n}(x_0y_i+y_0x_i)\varepsilon_i+uv\varepsilon_k \end{aligned} \tag{3.14}$$

The amount of the introduced error by this operation equals uv which makes an impact on the result if $\sum_{i=1}^{n} \mid x_i \mid$ or $\sum_{i=1}^{n} \mid y_i \mid$ becomes as large as x_0 and y_0 respectively.

3.4.4 Avoiding Repeated Evaluation

Each non-affine operation on affine forms produces one new error variable that is treated as if it was totally independent of all other existing sources of uncertainty. But fortunately the new error may cancel out in part by further operation on the same result.

The somewhat silly calculation of

$$(1.2-0.3\varepsilon_1)(2+0.5\varepsilon_2)-(1.2-0.3\varepsilon_1)(2+0.5\varepsilon_2) \tag{3.15}$$

[1] More precise bounds can be estimated using more complex formulas

equals to zero if first the product $(1.2 - 0.3\varepsilon_1)(2 + 0.5\varepsilon_2) = 2.4 + 0.6\varepsilon_1 + 0.6\varepsilon_2 + 0.15\varepsilon_3$ is calculated. If Formula 3.15 is first splitted into two products two new error variables ε_3 and ε_4 are generated; this way results in $0.15\varepsilon_3 - 0.15\varepsilon_4 = [-0.3, 0.3] \neq 0$.

Note that it is extremely important to avoid repeated computation of the same term, not only to save computational costs but to minimize the loss of constraints between the error variables which may increase the accuracy of the computation.

3.5 Results

Affine arithmetic is an approach to calculate formulas on intervals more precisely than standard interval arithmetic does. Of course it comes along with higher computational costs and a higher demand on storage. But the increase of accuracy may be worth the effort.

Affine arithmetic has already applied successfully within computer graphics e.g. to enumerate implicit surfaces [5], to calculate surface intersection [4], to evaluate procedural shaders over areas [11], and to ray-tracing of displacement shaders [10]. These are applications where a parametrized surface or function is subdivided, using AA to compute bounds on quantities that influence the refinement decision. Therefore AA seems to be suitable to use in a hierarchical radiosity algorithm too.

Chapter 4

Bounded Radiosity with Affine Arithmetic

Outgoing from Stamminger's bounded radiosity algorithm this thesis tries to apply affine arithmetic in all situations where non-exact quantities are computed. This includes finding an appropriate representation for the location of surfaces or clusters and their normals, computing bounds on the form factor by analyzing the geometric properties between two objects, making the decision on subdividing the surfaces and performing the push/pull operation on gathered radiosity. Even the applicability of AA to visibility tests is investigated.

4.1 Intervals, Affine Forms and Typical Values

One difficulty in translating the algorithm to use with AA has been that the independent treatment of upper and lower bound, as it is allowed within standard interval arithmetic, cannot be transfered to affine arithmetic without loss of all error variables and thus all advantages of the affine arithmetic approach. Especially when the domain of some quantity is limited by nature (e.g. the form factor between two patches may never exceed one) also the calculated bounds should be adjusted to avoid unnecessary wide intervals. During the push/pull operation the minimum and maximum value of the gathered radiosity in one branch of the hierarchy has to be determined and propagated. This really is not an affine operation. A one to one translation of intervals into affine forms is not always possible. Sometimes the affine form degrades to represent a simple interval[1] and sometimes the way of computation must change to suite affine forms.

Since today's graphics hardware and most software expect exact real numbers and not intervals to represent colors, there must be a step where the information of the interval is thrown away and a single value within its bound is determined as a typical value for the interval. Because an exact value of the radiosity cannot be calculated the choice of the typical

[1] Also storage space can justify the reduction of an affine form to represent its corresponding interval.

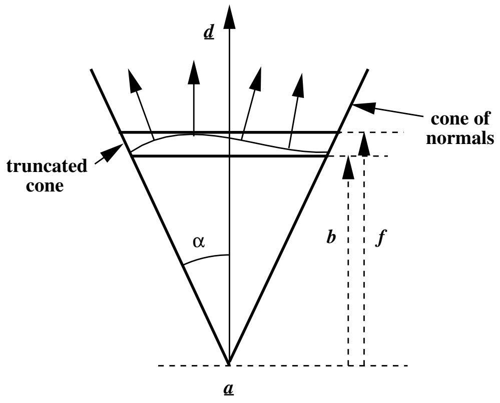


Figure 4.1: Anchored cone of normals

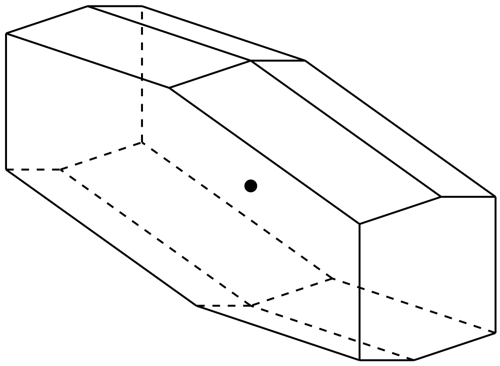


Figure 4.2: A random 3D affine form

value is somewhat arbitrary. Nevertheless there exist several ways to find a representative approximation.

4.2 Describing Geometry with AA

Instead of generating the anchored cone of normals (see [20]) it is more appropriate to represent the spatial extent and the normals of some object with three-dimensional affine forms if they are computed and further processed using affine arithmetic.

4.2.1 Cone of Normals vs. Three-dimensional Affine Forms

Each anchored cone of normals is defined by five parameters, see Figure 4.1: The "main" direction of the surface normals yields the axis $\underline{d}$ of the cone, and the cone angle α bounds the variation of the normals over the object. The cone attached to the apex $\underline{a}$ now contains all normals. With further parameters b and f a truncated cone can be defined that bounds the spatial extent of the object.

A three-dimensional vector consisting of affine forms $\underline{\hat{v}} = (\hat{x}, \hat{y}, \hat{z})^T$ constrains the values of the quantity $\underline{v}$ to lie in a point-symmetric convex 3D projection of an n-dimensional hypercube where n is the number of distinct error variables of $\hat{x}$, $\hat{y}$ and $\hat{z}$. Its center is given by $(x_0, y_0, z_0)^T$. The three components may share some error variables and thus are partly dependent on the same sources of error. This improves a 3D-affine form against a simple bounding box $([x_{\min}, x_{\max}], [y_{\min}, y_{\max}], [z_{\min}, z_{\max}])^T$ where the three intervals are implicitly independent. A random vector of affine forms is depicted in Figure 4.2.

Even if 3D affine forms cannot take arbitrary shapes due to their point-symmetrical structure they may fit the bundle of normals over some surface more precisely than an anchored cone of normals. As their name depicts, cones of normals represent the variation of the nor-

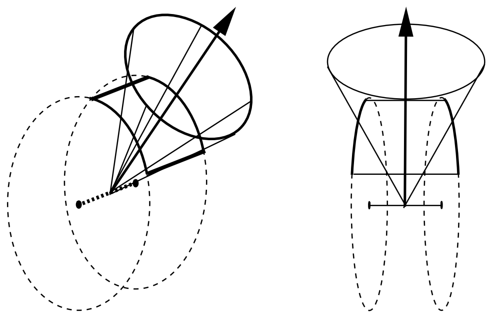


Figure 4.3: Normals bounded by a cone of normals ...

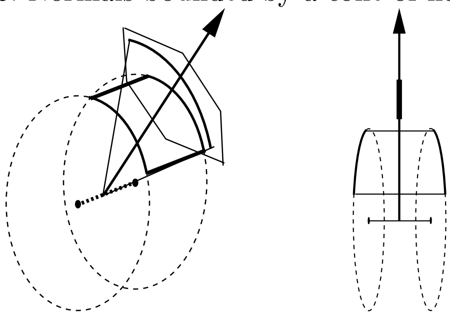


Figure 4.4: ... and a 3D affine form

mals by a cone. Variation of normals in only one plane results in a whole cone that is to say variation in all planes that are rotated about the cone's axis. This is depicted in Figure 4.3 and Figure 4.4 where the normals of an section of a cylinder are first bounded by a cone of normals and then by an affine form.

One advantage of the representation using cones of normals lies in the fact that the main direction may be normalized without greater effort. Because of this the calculation of the cosine between two cones is straight forward. This is not true for 3D affine forms because those objects normally contain vectors of different lengths (see Subsection 4.4.4).

Using affine arithmetic two affine forms are necessary to describe the whole geometry of a patch; one for the spatial extent and one for the normals. If they are used more than once in some computational chain one must ensure that identical affine forms are used. Recomputation would produce two different forms.

4.3 Bounding Normals and the Spatial Extent

For each kind of surface the bounding 3D affine form has to be computed differently to best fit the surface. As one might expect, a better approximation than an axis-aligned bounding box exists for nearly all kinds of surfaces but sometimes the complexity or the computational costs forbid a more precise approximation.

4.3.1 Bounds on Rectangular Parametrized Surfaces

One common class of surfaces is formed by rectangular parametrized surfaces. For each point on the surface exists one pair of orthogonal parameter values. To put it another way, given the parameters the coordinates of the corresponding point in space can be computed unambiguously according to the kind of surface. Ranges of parameters this way exactly describe parts of the surface. This is used for subdivision in particular. Subdividing the parameter space means subdividing the surface.

A representation of a rectangular parametrized surface using 3D affine forms is obtained as follows: Assume, the subdivided patch is given by two parameter intervals $[x_{\min}, x_{\max}]$ and $[y_{\min}, y_{\max}]$. These are transfered into affine forms $\widehat{x}$ and $\widehat{y}$ respectively. The same way coordinates are computed from concrete parameters, a 3D affine form is generated if affine arithmetic is applied and $\widehat{x}$ and $\widehat{y}$ are used as parameters.

For example, a parallelogram spanned by the vectors $\underline{a}$ and $\underline{b}$ and some point $\underline{c}$ on it possesses the parameter representation $\underline{x} = \underline{a}p + \underline{b}q + \underline{c}$, with $(p, q) \in [0, 1] \times [0, 1]$. The bounding affine form results in $\widehat{\underline{x}} = \underline{a}\widehat{p} + \underline{b}\widehat{q} + \underline{c}$ with $\widehat{p} = 0.5 + 0.5\varepsilon_1$ and $\widehat{q} = 0.5 + 0.5\varepsilon_2$ which even is exact because the parameter representation only contains affine operations. No more than those points which are actually part of the parallelogram are included in the 3D affine form.

The part of a z-axis aligned cylinder of radius 1 centered at the origin with a z-value ranging from -1 to 2 ($\widehat{z} = 0.5 + 1.5\varepsilon_3$) and an opening angle $\alpha \in [0, \frac{\pi}{4}]$ ($\widehat{\alpha} = \frac{\pi}{8} + \frac{\pi}{8}\varepsilon_4$) is bounded by $\widehat{\underline{x}} = (\cos\widehat{\alpha}, \sin\widehat{\alpha}, \widehat{z})^T$ (see Figure 4.5) where $\cos\widehat{\alpha}$ and $\sin\widehat{\alpha}$ are the affine arithmetic counterparts to the cosine and sine. These operations introduce of course new errors due to their non-affine nature which even worse are independent in the x and y component. Among other things also spheres and other quadrics may be described with rectangular parameters and a bounding 3D affine form is obtained in a similar way.

Most of these parametrized surface also provide a formula to derive the normal at some surface point. This allows computation of a bounding 3D affine form for the normals the same way as for the spatial extent. Since a point and its corresponding normal depend on exactly the same parameter values one should use identical affine forms representing parameter intervals when computing both the spatial extent and the normals. The correlation between both quantities is preserved this way but to ensure the use of identical affine form parameters they either have to be stored, or points and normals have to be computed simultaneously even if only one set of them is needed. In order to avoid repeated evaluation they have to be cached which enlarges the needs for storage space but also speeds up the algorithm if the bounding affine form is used more than once.

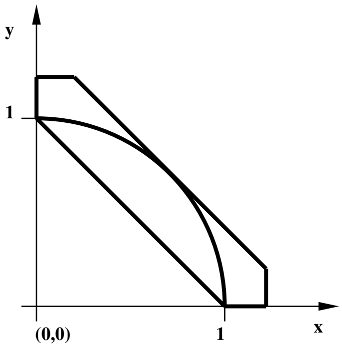


Figure 4.5: One part of a cylinder bounded by an affine form (top view)

4.3.2 Bounds on Triangles and Polygons

Other objects like triangles or other convex polygons do not possess an analytic representation depending on two independent parameters that allows to obtain an affine form representation this way. Fortunately, their normals are constant. For triangles an optimal bounding 3D affine form is given by a parallelogram spanned by two edges of the triangle or any other sheared parallelogram with boundaries defined by one edge and the remaining node of the triangle. But still the resulting expression contains twice as many points as the triangle does.

Construction of an optimal bounding 3D affine form for arbitrary convex polygons is far more complex. The best approximation in the sense of minimizing the maximum distance between some arbitrary point of the polygon and some of the 3D affine form is reached in three steps. First, find the center $\underline{c}$ of that surrounding circle with the smallest radius. All polygon points are mirrored about point $\underline{c}$ constructing a new convex hull, a point symmetric polygon which actually coincides with the corresponding affine representation. At last, the 3D affine form is given by $\underline{\widehat{p}} = \underline{c} + \frac{1}{2}\underline{a}_1\varepsilon_1 + \frac{1}{2}\underline{a}_2\varepsilon_2 + \cdots + \frac{1}{2}\underline{a}_{n/2}\varepsilon_{n/2}$ where $\underline{a}_1, \ldots, \underline{a}_n$ denote the edges of the constructed convex hull. Only $\underline{a}_1, \ldots, \underline{a}_{n/2}$, one half of them are sufficient because $\underline{a}_i = -\underline{a}_{n/2+i}$.

This sense of optimization sometimes produces unintended effects as in Figure 4.7 where the trapezoid is mirrored about its longest edge generating a hexagon and not a rectangle. The polygon's center of gravity may be another choice for point $\underline{c}$, and depending on the application even further optimizing points may be defined. But how should the center $\underline{c}$ be chosen to minimize the area of the 3D affine form, to minimize the sum $\sum_{i=1}^{n/2} \mid \underline{a}_i \mid$ or to minimize the maximum of the minimal distances between the polygon's and the the affine form's border. Finding $\underline{c}$ and constructing the new convex hull are obviously non-trivial tasks and in the worst case $\underline{\widehat{p}}$ may contain as many error variables as points defining the polygon raising the effort of further computations. This is why in this thesis' implementation a polygon is simply approximated by a 3D affine form of the bounding box and no enhanced technique is used.

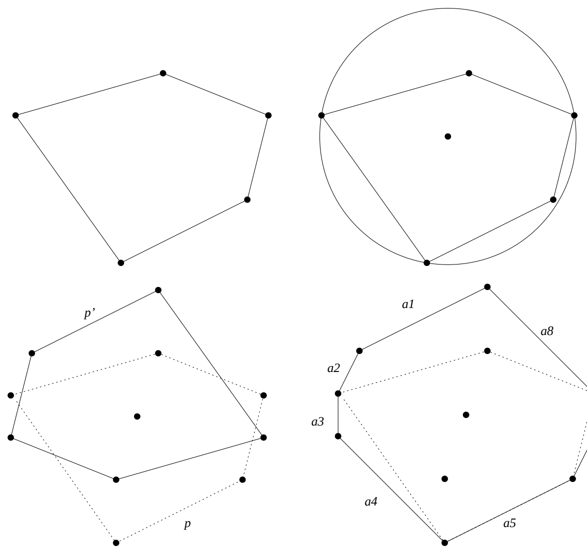


Figure 4.6: Constructing a bounding 3D affine form for convex polygons

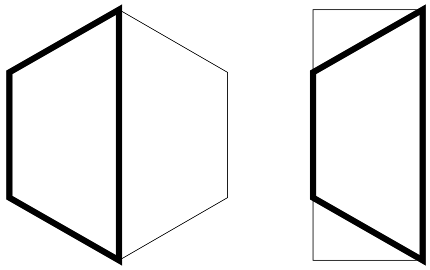


Figure 4.7: Possible bounding affine forms of a trapezoid

4.3.3 Bounds on Clusters

Collections of objects, clusters, suffer from similar problems like polygons. For simplicity assume that a cluster contains only two objects o_1 and o_2 with their bounding affine forms $\underline{\widehat{o}}_1$ and $\underline{\widehat{o}}_2$. Given the convex hull[2] of two objects — of course, the same applies to arbitrary shaped objects — a point symmetric convex hull may be constructed like for polygons: find an optimal center $\underline{c}$, optimal in some sense, and mirror all points about it. Unfortunately, the surface of the resulting hull consists of triangles. Because the surfaces of 3D affine forms are always build up by polygons with edges that are parallel in pairs no exact corresponding representation with affine forms exists and it is not quite clear how to construct an approximation that will fit best. Since an optimal bounding 3D affine form for arbitrary shaped clusters or objects is hard to determine other approximations have to be found.

Intuitively, a bounding 3D affine form for two objects can be calculated by simply blending

[2]Appendix C describes one way to determine the convex hull of arbitrary 3D affine forms

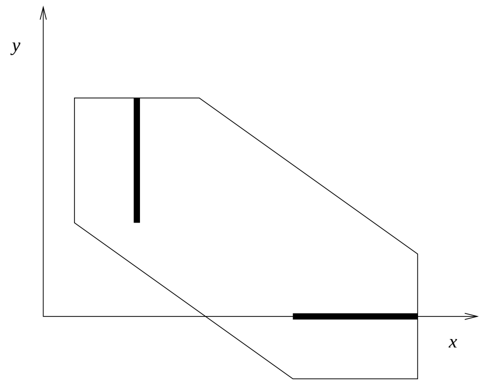


Figure 4.8: Some cluster for two lines generated by blending

between $\underline{\widehat{o}}_1$ and $\underline{\widehat{o}}_2$ with some unknown value $\widehat{t} = 0.5 + 0.5\varepsilon_t \widehat{=} [0, 1]$:

$$
\begin{aligned}
\underline{\widehat{x}} &= \underline{\widehat{o}}_1 \widehat{t} + \underline{\widehat{o}}_2 (1 - \widehat{t}) \\
&= \frac{1}{2} (\underline{\widehat{o}}_1 + \underline{\widehat{o}}_2 + \varepsilon_t (\underline{\widehat{o}}_1 - \underline{\widehat{o}}_2))
\end{aligned} \tag{4.1}
$$

The first term $\frac{1}{2}(\underline{\widehat{o}}_1 + \underline{\widehat{o}}_2)$ is computed exactly with affine arithmetic because only affine operations are used but the single multiplication of two affine forms ε_t and $(\underline{\widehat{o}}_1 - \underline{\widehat{o}}_1)$ results in independent errors in the x, y and z component of $\underline{\widehat{x}}$. Because $\underline{\widehat{o}}_1$ and $\underline{\widehat{o}}_2$ normally do not share any error variable the introduced error equals the amount of uncertainty of the term $\frac{1}{2}(\underline{\widehat{o}}_1 + \underline{\widehat{o}}_2)$ in each component; but no correlations exist between them. 3D affine forms generated this way tend to get too wide to be useful.

In Figure 4.8 the cluster for two lines $\underline{\widehat{l}}_1 = (1.5, 3+\varepsilon_1)^T$ and $\underline{\widehat{l}}_2 = (5+\varepsilon_2, 0)^T$ is constructed $\underline{\widehat{c}} = (3.25 + 0.5\varepsilon_2 - 1.75\varepsilon_t + 0.5\varepsilon_3, 1.5 + 1.2\varepsilon_1 + 1.5\varepsilon_t + 0.5\varepsilon_4)^T$. Note the unnecessary overshoot in both directions. This overshoot increases when more than two objects are combined into one cluster. In addition, the constructed 3D affine form contains every error variable of each included object, a considerable problem for clusters of the entire scene.

Due to the absence of better and useful approximations the axis aligned bounding box is an unprecise but compact and simple to construct alternative to deal with. Since each surface provides an affine form bounding its normals there is only one difference in the treatment of the spatial extent and the normals when constructing some cluster. Normals have to be normalized afterwards. Table 4.1 compares clustering using bounding boxes to clustering using bounding 3D affine forms generated by blending. Given the same error threshold, the second alternative produces more links and needs even more time to compute the clusters' form factor.

[3] See Chapter 6 for a scene description.

Scene[3]	Norm	Iterations	Oracle Threshold	Bounding Box		Blending	
				Links	Time	Links	Time
C	L_2	5	0.001	622561	370s	776367	489s
E	L_1	5	0.07	603614	685s	687669	811s

Table 4.1: Links generated by representing clusters with bounding boxes or 3D affine forms constructed by blending.

4.4 The Form Factor

As denoted in Section 2.7 the area and the bounds on the radiosity of the sender, as well as bounds on the form factor kernel between two interacting patches are sufficient to bound the transported radiosity. Instead of standard interval arithmetic now affine arithmetic is applied to the formula to compute those bounds.

4.4.1 Bounds on the Form Factor Kernel

Starting from Equation 2.11 which is replicated here,

$$k_{ij}(\underline{x}, \underline{y}) = \frac{\cos\theta_x \cos\theta_y}{\pi r^2} V(\underline{x}, \underline{y}) \tag{4.2}$$

an affine arithmetic formulation including the product with the area A_i may be found. Each term suffering from some uncertainty is represented as an affine form:

$$\widehat{f}_{ij}(\underline{\widehat{x}}, \underline{\widehat{y}}) = A_i \frac{\widehat{\cos\theta_x}\widehat{\cos\theta_y}}{\pi \widehat{r}^2} \widehat{V}(\underline{\widehat{x}}, \underline{\widehat{y}}) \tag{4.3}$$

The bounding 3D affine forms $\underline{\widehat{x}}$ and $\underline{\widehat{y}}$ allow to bound all possible directions between their two patches $\underline{\widehat{d}} = \underline{\widehat{y}} - \underline{\widehat{x}}$. Assume $\underline{\widehat{d}}'$, $\underline{\widehat{n}}_x$ and $\underline{\widehat{n}}_y$ to be the normalized direction and normals respectively. Equation 4.3 results in

$$\widehat{f}_{ij}(\underline{\widehat{x}}, \underline{\widehat{y}}) = A_i \frac{< \underline{\widehat{n}}_x \mid \underline{\widehat{d}}' >< \underline{\widehat{n}}_y \mid -\underline{\widehat{d}}' >}{\pi ||\underline{\widehat{d}}||^2} \widehat{V}(\underline{\widehat{x}}, \underline{\widehat{y}}). \tag{4.4}$$

The equivalent formula

$$\widehat{f}_{ij}(\underline{\widehat{x}}, \underline{\widehat{y}}) = A_i \frac{< \underline{\widehat{n}}_x \mid \underline{\widehat{d}} >< \underline{\widehat{n}}_y \mid -\underline{\widehat{d}} >}{\pi ||\underline{\widehat{d}}||^4} \widehat{V}(\underline{\widehat{x}}, \underline{\widehat{y}}) \tag{4.5}$$

where the normalization of $\underline{\widehat{d}}$ is taken out of the dot product tends to produce less precise results because the term $\frac{1}{r^4} = \frac{1}{(r^2)^2}$ now consists of three non-affine operations and the multiplications in the dot product introduces larger error if $||\underline{\widehat{d}}|| > 1$.

Note that the single terms of these equation are combined exclusively by non-affine operations each of them introducing a new source of error that will never cancel out. Grouping

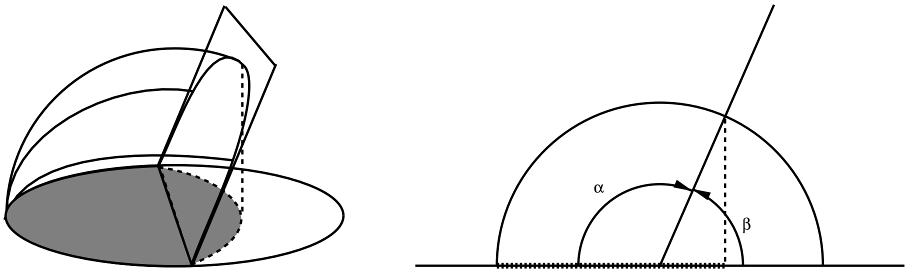


Figure 4.9: Bounding the form factor with Nusselt's analogy.

the terms in different ways does not only affect the computational time but also the precision. Each of the involved terms of Equation 4.4 and its influence on the formula is now investigated in more detail and several ways of gaining the corresponding affine forms are discussed.

4.4.2 The Square of the Distance

In reality two neighboring surfaces may touch each other at some common boundary. Bounding affine forms of two objects even may penetrate each other if the objects do not intersect. In both cases the minimal calculated distance between these object equals to zero; f_{ij} becomes singular.

One way to redress this problem of these natural singularities for convex objects is given by Nusselt's analogy. Assume $\underline{p}$ to be some point of the common boundary and Ω the hemisphere over the receiver centered at $\underline{p}$. Since both surfaces are convex, a plane through $\underline{p}$ can be determined that divides the hemisphere and the space above it into two parts in such a way that the sender is guaranteed to be totally contained in only one of them (see Figure 4.9). The projection of the occluded part of the hemisphere onto its base plane is proportional to the maximum value of the form factor between the two patches. Conservatism is ensured if the bounding plane and the receiver enclose the smallest angle between the sender and receiver, that is when the dot product of the normals gets minimal. It is easily seen that the maximum form factor is given by

$$\begin{aligned} f_{ij\,\max} &= \frac{1}{2}(1+\cos\beta) = \frac{1}{2}(1-\cos\alpha) \\ &= \frac{1}{2}(1- < \underline{\widehat{n}}_i \mid \underline{\widehat{n}}_j >_{\min}). \end{aligned} \tag{4.6}$$

For the lower bound no better estimate exists than $f_{ij\,\min} = 0$. Affine arithmetic is only used to bound the dot product, and a new interval is constructed by independent upper and lower bounds. All dependency of the form factor on the underlying patches is implicitly lost this way but obviously, this result is more valuable than no result.

Using affine arithmetic another kind of singularity came up although neither the surfaces nor their bounding 3D affine forms intersect. The definition of the affine approximation of

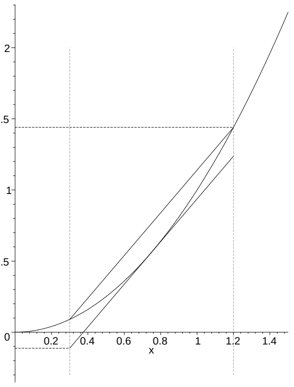


Figure 4.10: Chebyshev approximation: $(0.75 + 0.45\varepsilon_1)^2 \approx 0.66 + 0.68\varepsilon_1 + 0.1\varepsilon_2 \widehat{=} [-0.11, 1.44]$

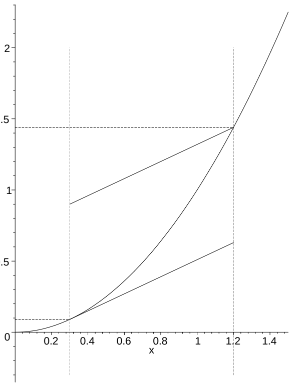


Figure 4.11: Min-range approximation: $(0.75 + 0.45\varepsilon_1)^2 \approx 0.77 + 0.27\varepsilon_1 + 0.41\varepsilon_3 \widehat{=} [0.09, 1.44]$

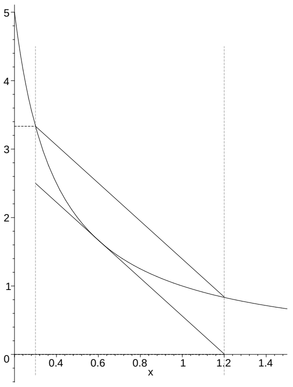


Figure 4.12: Chebyshev approximation: $1/(0.75 + 0.45\varepsilon_1) \approx 1.67 - 1.25\varepsilon_1 + 0.42\varepsilon_4 \widehat{=} [0, 3.33]$

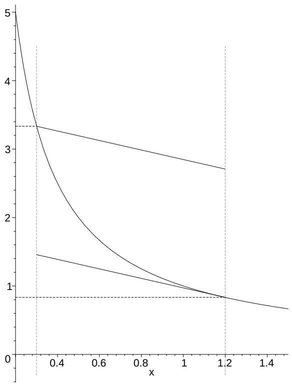


Figure 4.13: Min-range approximation: $1/(0.75 + 0.45\varepsilon_1) \approx 2.08 - 0.31\varepsilon_1 + 0.94\varepsilon_5 \widehat{=} [0.83, 3.33]$

the function $x \to x^2$ that minimizes the maximum error causes these anomalies. Due to this Chebyshev sense of optimization $(\widehat{x}^2)_{\min}$ may take values less than $(\widehat{x}_{\min})^2$. Especially for wide $\widehat{x}$ with a small lower bound this may result in possibly negative values for $\widehat{x}^2$ and the term $\frac{1}{\widehat{x}^2}$ is no longer defined.

Another kind of optimization is more appropriate in this situation; that is minimizing the width of the computed interval (min-range approximation [22]). The affine approximation of $x \to x^2$ must ensure that $(\widehat{x}^2)_{\min} = (\widehat{x}_{\min})^2$ and $(\widehat{x}^2)_{\max} = (\widehat{x}_{\max})^2$. The influence or the coefficients of the error variables of the parameter $\widehat{x}$ in the result of course is reduced this way, and a larger error is produced; but in the special case of a square as a denominator

Scene[5]	Norm	Iterations	Oracle Threshold	generated Links		
				Case 1	Case 2	Case 3
A	L_2	1	0.0001	659321	652993	937833
C	L_2	5	0.001	622561	623437	764697
D	L_1	5	0.01	428694	431771	442313
E	L_1	5	0.07	557437	557489	573993

Table 4.2: Number of links generated with or without applying Nusselt's analogy and the Chebyshev approximation of $x \to \frac{1}{x}$

the overall result becomes more precise. Tests showed that it is sometimes useful to calculate $x \to \frac{1}{x}$ the same way within the form factor formula.

Table 4.2 lists the number of links that are created in the three[4] following cases for several scenes with otherwise unchanged parameters:

1. Min-range approximation for $x \to \frac{1}{x}$ together with applying Nusselt's analogy.
2. Chebyshev approximation for $x \to \frac{1}{x}$ together with applying Nusselt's analogy.
3. Min-range approximation without applying Nusselt's analogy.

4.4.3 Bounds on the Projected Area

In Equation 4.3 the term $A_i\widehat{\cos\theta_x}$ denotes the projected visible area of the sender with respect to the direction of the receiver. Self-occlusion and curvature must be taken into account when the area of the sender is determined. The multiplication of the total area A_i times the dot product $< \underline{\widehat{n}_x} \mid \underline{\widehat{d}'} >$ may produce an overestimated upper bound. The lower bound by contrast is underestimated when the cosine is negative since an area is always positive; zero would be a better choice.

More precise upper bounds, particularly for clusters where the total area is computed as the sum of the areas of all contained objects, are found by the bounding 3D affine form[6]. Although the area of all surfaces in the cluster can become arbitrary large, its projected area is bounded by the area of the surrounding affine form. Because it is hard to calculate the exact surface area of a 3D affine form it is easier to bound it by the area of its projection onto the three coordinate planes. Area computation of 2D affine forms is explained in Appendix B. Viewing at some convex solid object the projected area does not exceed more than half of the surface area. Thus one half of the area computed this way yields an upper bound for the projected area of the object. In up to 50% of all interactions with sending clusters this

[4]The program did not work well if the Chebyshev approximation is used for $x \to x^2$. Thus no results for this case are represented here.

[5]See Chapter 6 for a scene description.

[6]Stamminger *et. al.* used the bounding box instead.

method yields a smaller upper bound than the dot product. But it is more time-consuming and does not reduce the number of generated links significantly.

Further, the 3D affine form may be projected directly with respect to the set of interacting directions $\widehat{\underline{d}}$. Firstly, two normalized orthogonal vectors $\widehat{\underline{p}}_1, \widehat{\underline{p}}_2$ have to be constructed that define some planes perpendicular to $\widehat{\underline{d}}$. The bounding 3D affine form $\widehat{\underline{x}}$ now is projected onto these planes in one step by computing the dot products $< \widehat{\underline{p}}_1 \mid \widehat{\underline{x}} >$ and $< \widehat{\underline{p}}_2 \mid \widehat{\underline{x}} >$. The area of the obtained 2D affine form bounds the projected area of the object. Note, constructing $\widehat{\underline{p}}_1$ and $\widehat{\underline{p}}_2$ and calculating the dot products needs a lot of multiplications, non-affine operations. The effort and the resulting error cause problems. In less than 2% of all cases better bounds are computed, but the overall result is hardly affected.

4.4.4 Normalizing Vectors of Affine Forms

The dot product of two vectors equals the cosine of the angle between them only if both vectors are normalized. Vectors of different length are included when normals are represented by their bounding 3D affine form. For clusters and for the difference $\widehat{\underline{d}}$ between two objects even the zero-vector may be contained. The standard way of normalizing some vector

$$\widehat{\underline{n}} = \begin{pmatrix} \widehat{x} \\ \widehat{y} \\ \widehat{z} \end{pmatrix}; \qquad \frac{\widehat{\underline{n}}}{\|\widehat{\underline{n}}\|} = \frac{1}{\sqrt{\widehat{x}^2 + \widehat{y}^2 + \widehat{z}^2}} \begin{pmatrix} \widehat{x} \\ \widehat{y} \\ \widehat{z} \end{pmatrix} \tag{4.7}$$

succeeds for most tight vectors and yields tight results but fails in the above cases due to the problems of $x \to x^2$ and $x \to \frac{1}{x}$ mentioned earlier. Even the min-range approximation of $x \to x^2$ and $x \to \frac{1}{x}$ does not solve the problem of normalizing a 3D affine form that contains the zero-vector. In addition the number of error variables will increase within each component.

Looking at the normalizing function of one component, for example x

$$f(\widehat{x}, \widehat{y}, \widehat{z}) = \frac{\widehat{x}}{\sqrt{\widehat{x}^2 + \widehat{y}^2 + \widehat{z}^2}} \tag{4.8}$$

a simplified function, plotted in Figure 4.14, is given by

$$f(\widehat{x}, c) = \frac{\widehat{x}}{\sqrt{\widehat{x}^2 + c}}, \qquad \text{with} \qquad c \geq 0. \tag{4.9}$$

Of course the dependencies of the resulting $\widehat{x}$ on the error variables of $\widehat{y}$ and $\widehat{z}$ is neglected this way but this function is much easier to control. Note that the function is center-symmetric.

Assume two positive values $c_1 < c_2$; it follows that $\frac{|\widehat{x}|}{\sqrt{\widehat{x}^2 + c_1}} \geq \frac{|\widehat{x}|}{\sqrt{\widehat{x}^2 + c_2}}$. For all values of the quantity $\widehat{c} = \widehat{y}^2 + \widehat{z}^2$ the function $f(\widehat{x}, \widehat{c})$ is bounded by

$$\frac{|\,\widehat{x}\,|}{\sqrt{\widehat{x}^2 + \widehat{c}_{\min}}} \geq \frac{|\,\widehat{x}\,|}{\sqrt{\widehat{x}^2 + \widehat{c}}} \geq \frac{|\,\widehat{x}\,|}{\sqrt{\widehat{x}^2 + \widehat{c}_{\max}}}. \tag{4.10}$$

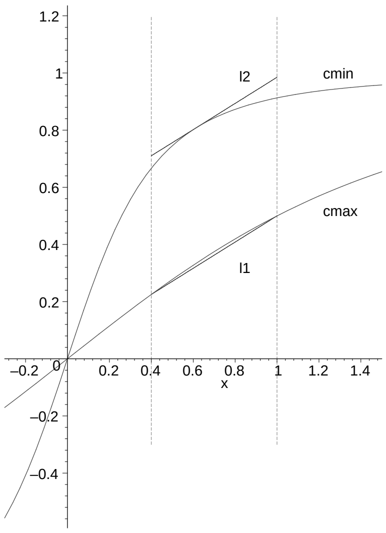


Figure 4.14: Bounding the normalizing function

$f(x, c_{\min})$ and $f(x, c_{\max})$ are upper and lower bounds on the function for $x > 0$.

Since the derivative of $f(\widehat{x}, c)$ results in

$$f'(\widehat{x}, c) = \frac{c}{(\widehat{x}^2 + c)^{3/2}} \tag{4.11}$$

f and f' are monotonic in $\widehat{x}$. The gap in the domain is closed by $f(0, c) = 0$.

When an affine approximation of $f(\widehat{x}, \widehat{c})$ is calculated, three distinct cases with respect to the parameter $\widehat{x}$ are possible. Let $a = \widehat{x}_{\min}$ and $b = \widehat{x}_{\max}$, the cases are $a \geq 0$ and thus $b \geq 0$; $a \leq 0$ and $b \leq 0$ which is symmetric to the first case, and $a < 0$ and $b \geq 0$.

In the first case, as in the second, two parallel lines l_1 and l_2 have to be found that bound the functions $f(x, c_{\min})$ and $f(x, c_{\max})$ over the interval $[a, b]$, like in Figure 4.14. Assume some α and some x_0 so that $f'(x_0, c_{\min}) = \alpha$ or $x_0 = \sqrt{\frac{c_{\min}}{\alpha}^{2/3} - c_{\min}}$, l_1 and l_2 are defined as

$$\begin{aligned} l_1(x) &= \alpha x + f(a, c_{\max}) - \alpha a \quad \text{and} \\ l_2(x) &= \alpha x + f(x_0, c_{\min}) - \alpha x_0. \end{aligned} \tag{4.12}$$

Further, the affine approximation of $f(\widehat{x}, \widehat{c})$ may look like this

$$\begin{aligned} \widehat{f}(\widehat{x}, \widehat{c}) &= \alpha\widehat{x} + \frac{l_2(0) + l_1(0)}{2} + \frac{l_2(0) - l_1(0)}{2}\varepsilon_f \\ &= \alpha\widehat{x} + \frac{f(x_0, c_{\min}) + f(a, c_{\max}) - \alpha(x_0 + a)}{2} \\ &\quad + \frac{f(x_0, c_{\min}) - f(a, c_{\max}) - \alpha(x_0 - a)}{2}\varepsilon_f \end{aligned} \tag{4.13}$$

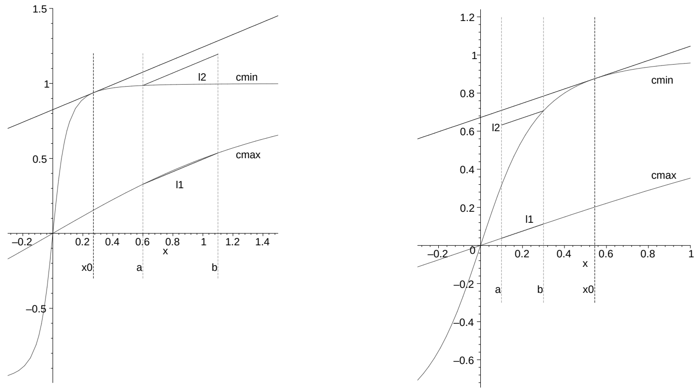


Figure 4.15: x_0 outside $[a, b]$

with some brand new error variable ε_f. A first choice for α should be

$$\alpha = \frac{f(b, c_{\max}) - f(a, c_{\max})}{b - a}. \tag{4.14}$$

If the corresponding x_0 is not contained in the interval $[a, b]$, this α minimizes the maximum error for all possible values of $\varepsilon_1, \ldots, \varepsilon_n$ of error variables in $\widehat{x}$ as depicted in Figure 4.15. For $x_0 \leq a$ it follows that $f'(x, c_{\min}) \leq \alpha$ for all $x \in [a, b]$. The minimum value of the maximum error is given by $\frac{1}{2}(f(a, c_{\min}) - f(a, c_{\max}))$, thus set $x_0 = a$. If $x_0 \geq b$, following the same arguments, the lower bound of the maximum error is $\frac{1}{2}(f(b, c_{\min}) - f(b, c_{\max}))$; choose $x_0 = b$. Variation of α between α and $f'(a, c_{\min})$ or α and $f'(b, c_{\min})$ respectively may reduce the average error without affecting the maximum error.

If $x_0 \in [a, b]$ that α that minimizes the maximum error cannot be obtained that simple. Variation of α about $\Delta\alpha$ yields some difference Δe in the error (see Figure 4.16).

$$\begin{aligned} \Delta e \quad = \quad & \underbrace{\mid \Delta\alpha \mid (b - a)}_{\Delta e_1} \\ & - \underbrace{(f(x_0, c_{\min}) + \alpha(b - x_0) - (f(x_1, c_{\min}) + (\alpha + \Delta\alpha)(b - x_1)))}_{\Delta e_2}, \end{aligned} \tag{4.15}$$

where $f'(x_1, c_{\min}) = \alpha + \Delta\alpha$ and $x_1 \in [a, b]$. Finding the minimal Δe for all possible $\alpha + \Delta\alpha \in [f'(b, c_{\min}), f'(a, c_{\min})]$ the maximum error is minimized by the corresponding $\alpha + \Delta\alpha$. Because normalizing vectors turned out to be a time consuming task this enhanced error reduction is not implemented.

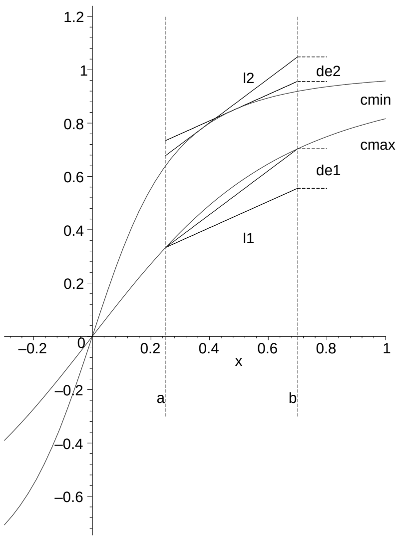


Figure 4.16: Minimizing the maximum error

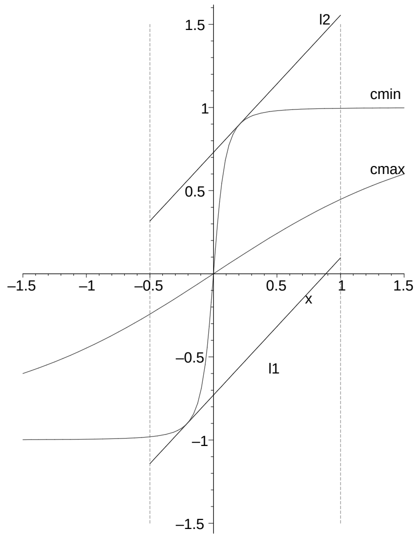


Figure 4.17: Bounding $f(\widehat{x}, \widehat{c})$ for $\widehat{x}_{\min} < 0$ and $\widehat{x}_{\max} > 0$

For the last case $a < 0$ and $b \geq 0$ no error minimization is performed either, although an optimal α can be found similar as before. As showed in Figure 4.17 one may determine the slope of the lines s_1 and s_2, $\alpha_1 = \frac{f(b,c_{\max})-f(a,c_{\min})}{b-a}$ and $\alpha_2 = \frac{f(b,c_{\min})-f(a,c_{\max})}{b-a}$. One of them has to be chosen as α, for example the smallest one. Because $f(x, c_{\min})$ is the upper bound for $x > 0$ and the lower bound for $x < 0$, two values $x_0 = +\sqrt{\frac{c_{\min}}{\alpha}^{2/3} - c_{\min}}$ and $x_1 = -\sqrt{\frac{c_{\min}}{\alpha}^{2/3} - c_{\min}}$ must be computed. If one of them lies outside the bounds $[a, b]$ it should be set to the nearest value. a now is exchanged by x_1 in the formula of l_1. The rest is defined as in the case above.

4.4.5 The Cosine of a Receiving Cluster

A cluster gathering radiosity from a range of interacting directions $\widehat{\underline{d}}$ must update the radiosity of its children during the push operation. Treating clusters as ordinary isotropic patches would result in adding the same amount of radiosity to the irradiance of each child no matter which orientation it possesses. Clearly, the surfaces in a cluster are not illuminated equally by some light source, some are back-facing and others are occluded.

This is why the value of the form factor where the cosine of the receiver is computed by $< \widehat{\underline{n}}_{\text{cluster}} \mid \widehat{\underline{d}}' >$, the product of the cluster's normal times the direction of the light source, is only used within the refinement decision. If the decision is negative and the energy transfer takes place across the current link in the hierarchy the form factor for each object in the cluster is recomputed with the receiver's cosine corresponding to the object, in order to gain the correct form factor. This is done by storing the form factor before multiplicating with the

View at a corner of Scene C using ...

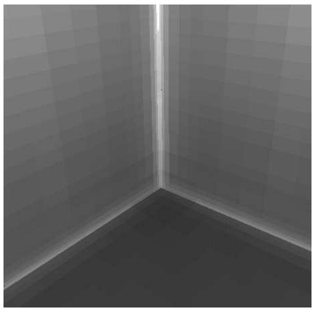


Figure 4.18: the center value, (404159 links)

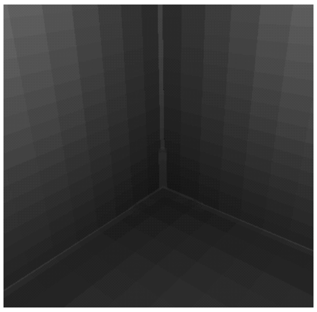


Figure 4.19: the typical value, (481862 links)

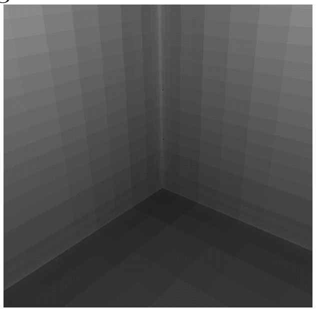


Figure 4.20: and a sampled value. (501099 links)

receiver's cosine of the cluster. This incomplete form factor is multiplied with the individual cosine afterwards. Stamminger [21], to the contrary, simply multiplies that form factor that is used for the refinement decision by 4 times the cosine of each child[7]. The first method increases the accuracy by consuming more storage space.

In [21] Stamminger proposed to apply a similar technique when a cluster acts as sender. If the cluster contains some light source severe visible artifacts emerge if the anisotropic nature of the cluster is neglected. The exitant radiance of the cluster with respect to the direction $\underline{\widehat{d}}$ will be better approximated if each child tests if it contributes to this radiance or if it is back-facing with respect to $\underline{\widehat{d}}$. A recursive test of course is very time consuming and thus the recursion is stopped at the next lower level.

4.4.6 A Typical Value for the Form Factor

Until now the form factor is bounded and represented by an affine form but after running a radiosity algorithm concrete values are commonly wanted. Instead of first computing affine forms for the radiosity of each patch and then after the algorithm has converged to pick out some random value out of the resulting radiosity interval, the visual occurrence will be significantly improved if the geometry properties of each single interaction is used to choose some typical value for the form factor that determines the amount of a typical radiosity interchange. The affine form still is needed for the refinement decision. To avoid arbitrary selection three different methods to gain a typical form factor value are given here:

- The simplest one is just to take the average value of the interval, $f_{typical} = \frac{\widehat{f}_{\max}+\widehat{f}_{\min}}{2}$. This choice minimizes the resulting maximum error $err = \max(f_{typical} - \widehat{f}_{\min}, \widehat{f}_{\max} - f_{typical}$ if $\widehat{f}$ is replaced by $f_{typical}$, and less subdivision is enforced. But it turns out that the average value seriously overestimates the real form factor near shared boundaries (see Figure 4.18).

[7] 1/4 is the average positive cosine value for randomly distributed surfaces in a cluster

- Another way is to derive the typical value for the form factor by calculating it from typical values of each term of Equation 4.3. The underlying affine arithmetic library automatically applies each operation simultaneously to both affine forms and their typical values. Note, that the typical value of some bounding 3D affine form again is an arbitrary choice when it is used to determine a form factor. The chosen point may be back-facing although most of the patch is front-facing. Due to its distance to the average value computing the typical form factor this way causes more subdivision but the artifacts of the first method are strongly reduced (see Figure 4.19).

- At last a typical value for the form factor may be approximated by sampling. This means additional effort whereas the values of the first two approaches come along with the standard way of computing the form factor interval. Accuracy is controlled by the number of rays shot, and thus it is a matter of time. The visual results of this method are still the best but it tends to enforce even more subdivision (see Figure 4.20).

4.4.7 Self-interaction

For non-convex objects, like clusters, we have to consider self-interaction, and thus a form factor describing this fact must be found. Stamminger suggested the following in [20] and [21]. The form factor for self-interaction is bounded by the interval $[0, 1 - cos^2\alpha]$ where α is the maximum deviation angle of the normals from the main direction.

A typical value for the form factor of the interaction of a patch with itself is computed last, after all other form factors are available. Let f_{sum} denote the sum of all typical form factors due to interaction with other objects gathered by the patch. Since one patch gathers radiosity and thus parts of its f_{sum} through all levels of the hierarchy the push/pull operation (Section 2.6) must be performed for f_{sum} the same way as for radiosity values. The idea is that all form factors of one patch in a closed environment sum up to 1. A typical value for the self-form factor thus is given by $f_{typical} = 1 - f_{sum}$. If the form factor sum of one patch exceeds 1 in a closed scene a correction method may be found by similar considerations.

4.5 Treatment of Radiosity Values

If radiosity values are represented by affine forms or intervals the pull operation as described in Section 2.6 must be changed slightly. The push operation can be adopted without any changes for surfaces and is performed as described in Section 4.4.5 for clusters.

During the pull operation only the typical value for the radiosity of some object can be computed by the area-weighted sum of the typical values of the children. To preserve conservatism, lower and upper bound must be treated another way. The minimum (and maximum) radiosity value of a parent object is given by the minimum (maximum) value of all its children because this child may be the only one of the entire object that is visible for some receiver.

4.5.1 Where to stop with affine arithmetic?

This brings up another question. Where to stop with affine arithmetic calculation? Since every error variable within an affine form slows down the computation, the affine form should be turned as soon as possible but as late as necessary into a simple interval with the loss of all error variables[8]. After the pull operation a new radiosity interval is constructed by the upper and lower bound. There is no way to save the initial error variables of the irradiance of the children.

The last step before the push/pull operation is the multiplication of the form factor with the radiosity of the sender. All radiosity values of one object gathered at the same level in the hierarchy share some error variables, those of the patch's normals and of its bounding 3D affine form. Only the coefficients of these error variables may cancel out if the affine form of the form factor is multiplied by the interval of the sender's radiosity with the result of a slightly tighter affine form for the entire gathered radiosity.

From another viewpoint it is preferable even to represent the form factor by a simple interval. The maximum range of the form factor by definition is $[0, 1]$. It does not make sense to allow the minimum and maximum value of the affine form representation to lie outside this interval because this would result in unnecessary imprecision or even negative radiosity values. In this thesis, after the form factor is computed with affine arithmetic, upper and lower bound are tested against $[0, 1]$ and a new interval is constructed with appropriate bounds.

4.5.2 Conservatism within one Iteration

No matter of how the irradiance interval is computed the resulting outgoing radiosity values of the patches in the lowest level of the hierarchy again are intervals. If they are used in the next iteration of the solution process the iterative method is not guaranteed to converge for the upper bounds [13]. At best, only far more subdivision will be enforced during the next iteration because not only the form factor is computed with some uncertainty but also all radiosity intervals in the next iteration step carry some error from the iterations before, and the multiplication of both results in error explosion. The influence of the uncertainty of the form factor is ignored in the following iteration step if the radiosity values of the leaves in each branch of the hierarchy are reduced to their typical values before the pull operation. One may choose if conservative bounds should be guaranteed during the entire computation with many more subdivided patches or if conservatism is ensured only for each iteration step, that is propagating the entire interval or only the typical value.

[8] Of course, one new error variable is constructed and to determine the range of an affine forms all coefficients must be collected. For affine form with many error variables this is a time consuming step.

4.6 The Oracle

The oracle is that part of a hierarchical radiosity algorithm that decides whether further subdivision is needed or the energy transport between two patches can be guaranteed to be approximated with sufficient precision if it takes place across the current link in the hierarchy. The subdivision criterion is tested at least once before the radiosity interchange is performed. It should be a low cost prediction whereas the form factor for the concrete energy exchange, the typical value, may be calculated with further effort including precise visibility computation afterwards.

If the typical value for the form factor is known in advance, before the refinement decision, the maximum error that results if the radiosity transported across a given link is approximated by $f_{typical}$ times the typical radiosity of the sender is given by

$$err = \max(\widehat{f}_{\max}\widehat{B}_{\max} - \widehat{f}_{typical}\widehat{B}_{typical}, \quad \widehat{f}_{typical}\widehat{B}_{typical} - \widehat{f}_{\min}\widehat{B}_{\min}). \tag{4.16}$$

This is significantly smaller than the conservative error for unknown typical values, that is $(f_{\max} - f_{\min})$ times the sender's radiosity

$$err = \max(\widehat{f}_{\max}\widehat{B}_{\max} - \widehat{f}_{\min}\widehat{B}_{typical}, \quad \widehat{f}_{\max}\widehat{B}_{typical} - \widehat{f}_{\min}\widehat{B}_{\min}). \tag{4.17}$$

Less subdivision is necessary to ensure the error to lie beneath the user defined error threshold if the typical value is known in advance.

4.6.1 Error Norms

Like in [21] and [13], one may choose from three different error norms: L_∞ that uses simply the above error, L_1 the error times the area of the receiver and L_2 that may be approximated by the error times the size of the receiver, for instance the radius of the bounding sphere. The L_1-norm is less sensitive to shadow boundaries because larger radiosity ranges are allowed for small patches. The L_2 and L_∞-norms improve this behavior at the cost of increased computation time. Shadow boundaries may be approximated even better if the user defined threshold controlling the refinement decision is lowered by some factor if partial visibility (see Section 4.7) is determined between the current pair of objects.

4.6.2 Subdividing Sender or Receiver

After the oracle has decided to refine one of the participating objects the sender or the receiver is subdivided depending on which subdivision will result in the largest error reduction. These decisions are of heuristic nature since the true change of error can only be estimated.

Lischinski *et al.* [13] proposed to subdivide the receiver if

$$(\widehat{f}_{\max} - \widehat{f}_{\min})\widehat{B}_{\max} \geq \widehat{f}_{\max}(\widehat{B}_{\max} - \widehat{B}_{\min}), \tag{4.18}$$

Scene[9]	Norm	Iterations	Oracle Threshold	Links	
				$A_i > A_j$	$A_i(0.3 + \Delta\cos_i)$ $> A_j(0.3 + \Delta\cos_j)$
A	L_2	1	0.001	46825	49649
B	L_2	1	0.0001	12425	12425
C	L_2	5	0.001	650493	622561
E	L_1	5	0.07	606598	573993

Table 4.3: Links generated using different decisions on subdividing sender or receiver

that is when the error due to a non-constant form factor is greater than the error due to non-constant radiosity of the sender. The sender is subdivided otherwise. But sources of constant radiosity will never by subdivided this way.

In addition the differences of the sender's and the receiver's geometrical properties are totally ignored. In Equation 4.3, neglecting the visibility, the range of three terms are responsible for the error of the form factor: the difference vector and the two cosine terms. The area of an object has influence on both the range of the distance and indirectly on the range of the cosine terms. The range of the normals of course affects the accuracy of the cosines. The maximum angle between all normals of one object would be a good measure for this but it is difficult and too expensive to obtain. Instead the cosine terms themselves may be used although other quantities than the range of the normals have influence on them too. For instance, the receiver j may be subdivided if

$$A_j(\cos_{j\,\max} - \cos_{j\,\min}) \geq A_i(\cos_{i\,\max} - \cos_{i\,\min}) \tag{4.19}$$

This criterion in return neglects non-constant radiosity of the sender. Even better subdivision will be obtained if the influence of the range of the cosine is lowered by some number, i.e. 0.3 turned out to be a good choice:

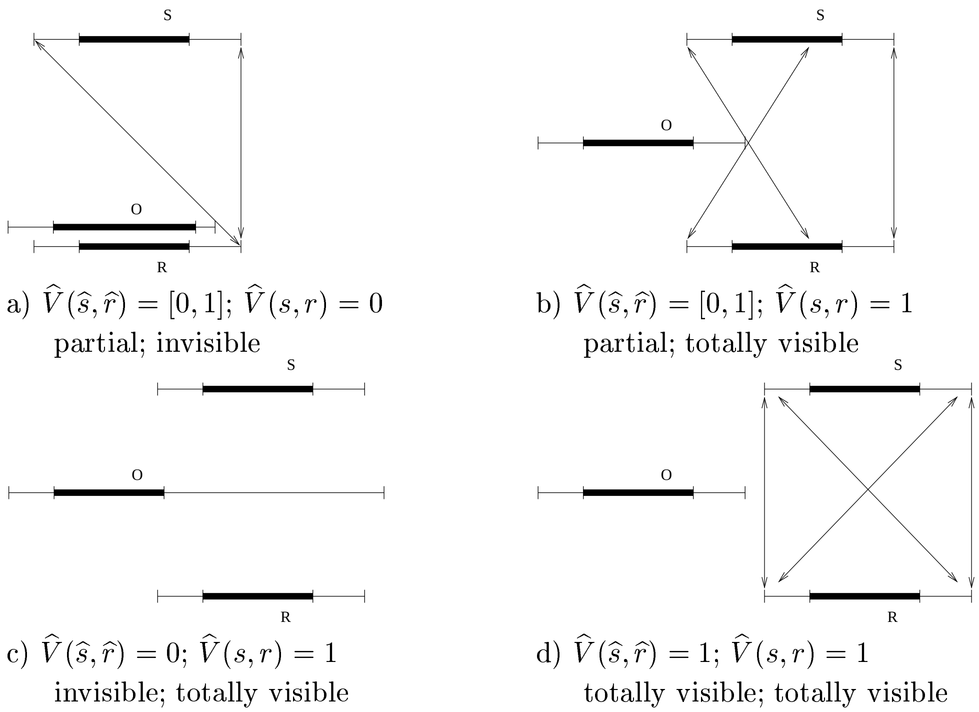
$$A_j(0.3 + \cos_{j\,\max} - \cos_{j\,\min}) \geq A_i(0.3 + \cos_{i\,\max} - \cos_{i\,\min}) \tag{4.20}$$

In Table 4.3 it can be seen that this criterion results in less links for scenes with curved surfaces or clusters than the criterion $A_i > A_j$.

4.7 Visibility Computation

The only term of Equation 4.3 not yet discussed is the mutual visibility $\widehat{V}(\underline{\widehat{x}}, \underline{\widehat{y}})$ denoting the fraction of one surface visible by the other. With respect to this fraction visibility may be classified into three categories: totally visible ($V = 1$), totally invisible ($V = 0$) or partially visible ($V \in [0, 1]$). Each surface in the environment may potentially act as an occluder between $\underline{\widehat{x}}$ and $\underline{\widehat{y}}$ and normally must be tested. Fortunately the hierarchical structure of the scene allows

[9]See Chapter 6 for a scene description.



a) $\widehat{V}(\hat{s},\hat{r}) = [0,1]$; $\widehat{V}(s,r) = 0$
partial; invisible

b) $\widehat{V}(\hat{s},\hat{r}) = [0,1]$; $\widehat{V}(s,r) = 1$
partial; totally visible

c) $\widehat{V}(\hat{s},\hat{r}) = 0$; $\widehat{V}(s,r) = 1$
invisible; totally visible

d) $\widehat{V}(\hat{s},\hat{r}) = 1$; $\widehat{V}(s,r) = 1$
totally visible; totally visible

Figure 4.21: Due to overshoot (thin lines) classification of visibility between 3D affine forms $\widehat{V}(\hat{s},\hat{r})$ and that of their real objects (thick lines) $\widehat{V}(s,r)$ diverges if they are partially visible or totally invisible.

to reduce these visibility tests. If one cluster does not intersect any of the connecting rays between $\underline{\hat{x}}$ and $\underline{\hat{y}}$ no surface in it will do either. Teller and Hanrahan [23] published an algorithm that classifies the visibility between each pair of objects in a scene in a preprocessing step that further reduces the number of possible occluders. But this method is beyond the scope of this thesis and does not work for all kinds of surfaces.

Instead, during runtime of the solution process of the radiosity equation for each processed pair of patches first, before the oracle is entered, the visibility is roughly approximated and conservatively classified. This means that if total visibility or invisibility cannot be guaranteed, the objects are classified as partially visible. After the refinement decision, if the energy interchange is enforced between the current objects sampling is used to determine a typical value for $\widehat{V}(\underline{\hat{x}},\underline{\hat{y}})$ that approximates the real value with higher precision.

4.7.1 Visibility Tests with Affine Arithmetic

The application of affine arithmetic to visibility tests has also been investigated but it turned out that it does not seem to be suitable for this task. The representation of surfaces with 3D affine forms already restricts the soundness of predicates produced using affine arithmetic for visibility computations. Because of the overshoot of 3D affine forms, as of all other representations that do not fit the enclosed object exactly, classification of the visibility between affine forms diverges from the classification of the real objects. In Figures 4.21a to 4.21c the affine representation wrongly classifies the real visibility. Only if total visibility is determined the real objects are guaranteed to be totally mutual visible (Figure 4.21d). Using affine arithmetic all other cases have to be classified partially visible $\widehat{V}(\widehat{x}, \widehat{y}) = [0, 1]$ since this is always true though imprecise. A quantitative typical value for the visibility cannot be determined using the 3D affine forms.

In order to perform the visibility test one has to test for all possible rays between the objects if they are intersected by some occluder. Usually, only a finite number of sample rays are shot and small occluders may be missed depending on the rays' distribution. Using affine arithmetic, the idea is that a 3D affine form is constructed that bounds all possible rays. For each potential occluder just one penetration test with the bounding affine form is needed instead of one test for each sample ray. This reduces not only the number of necessary tests but also produces conservative results.

An intersection test for 3D affine forms may look as easy as this: $\underline{\widehat{o}}_1$ and $\underline{\widehat{o}}_2$ intersect each other if the minimum distance between them is zero; $\mid \underline{\widehat{o}}_1 - \underline{\widehat{o}}_2 \mid_{\min} = 0$. Unfortunately all those objects will fail this intersection test that share a common boundary as those of which only the bounding affine forms and not the actual objects penetrate each other.

Another problem is to bound all connecting rays between the sender and the receiver. This is identically the same problem as finding the best approximation of the convex hull of two objects in Section 4.3. The quality of this approximation will have deep impact on the number of objects reported as occluders although they actually do not intersect any ray. Because no way of calculating good bounding affine forms for two arbitrary objects has been found and because of the shortcomings of the intersection test total visibility will be reported for only a small fraction of all totally visible pairs.

Other approaches that investigate the intersection of the vectors $(\underline{\widehat{s}} - \underline{\widehat{r}})$ and $\widehat{t}(\underline{\widehat{occ}} - \underline{\widehat{r}})$, where $\widehat{t}$ scales the difference $(\underline{\widehat{occ}} - \underline{\widehat{r}})$ in such a way that $\widehat{t}(\underline{\widehat{occ}} - \underline{\widehat{r}})$ is ensured to intersect $(\underline{\widehat{s}} - \underline{\widehat{r}})$ if $\underline{\widehat{occ}}$ intersects any ray, did not succeed either. The number of pairs classified totally visible is so small, in fact less than 1% of the totally visible connections are correctly classified, that it does not make sense to apply any of these visibility tests that are based on affine arithmetic. The causes are overshoot of the bounding 3D affine forms and errors generated by scaling (multiplication) of 3D affine forms by other affine forms.

Chapter 5

Implementation

The implementation of the described algorithm, combining the bounded radiosity method with affine arithmetic, is based on two given programs: an implementation of bounded radiosity within the Vision-project and a library on affine arithmetic providing different classes to represent color, points and single quantities with affine forms as well as basic arithmetic operations on them. The first has been written by Marc Stamminger and the second by Wolfgang Heidrich.

5.1 The Affine Arithmetic Library

The library that implements affine forms for different use and basic arithmetic operations is implemented in C++. Relevant for this thesis are four classes and the member functions described in the following.

5.1.1 AAScope

Each construction of a new interval and each non-affine operation on affine forms generates at least one new error variable that must be distinct from all other yet existing. An object of this class after construction ensures the uniqueness of generated error variables within its scope. It can be asked to create a totally new variable by newErrorVar() that returns a unique number. For independent calculations, two or more AAScope-objects may be instantiated providing distinct scopes. The currently active scope is stored in a static member variable and will be returned by a call to getActiveScope().

5.1.2 AAScalar

This class represents an affine form together with a scalar for its typical value. Each operation on an AAScalar object is automatically but separately performed on both. The center value of an affine form that is given by $\frac{\widehat{f}_{\max}-\widehat{f}_{\min}}{2}$ may differ from the typical value of the AAScalar object since it is treated independently of the affine arithmetic computation. As the error variables of an AAScalar and thus the object itself are valid just within one scope, a pointer to the corresponding AAScope object is stored with each AAScalar. Objects of different scopes may not be combined. An AAScalar contains additional information about the validity of the creating operation besides the error variables, the coefficients, the center and the typical value.

The important public methods are:

- AAScalar(scalar value= 0.0)
  AAScalar(scalar min, scalar max)
  AAScalar(scalar min, scalar max, scalar typical):
  The constructor of the class creates a new affine form with the range $[\min, \max]$ or value. If no explicit typical value is given it is set to the center of the interval.
- void getInterval(scalar& min, scalar& max):
  computes the range of the affine form and returns the minimum and maximum value.
- void getCenterError(scalar& center, scalar& error):
  also computes the range of the affine form but returns the center value and half the interval width.
- scalar getTypical():
  returns the typical value of this instance.

All basic operations on affine forms like =, +, -, *, /, sqr(), sqrt(), $\cos, \sin, \tan, \ldots$ are implemented. Those combining two quantities also are accessible as assignment-operators.

Changes on AAScalar

Three methods have been added to the class AAScalar that are needed to implement parts of this thesis:

- friend scalar area(AAScalar &x, AAScalar &y):
  This function calculates the upper bound on the area of the range of the two-dimensional affine form $(\widehat{x}, \widehat{y})$ as it is reported in appendix B.
- friend AAScalar convHull(AAScalar &a, AAScalar &b, scalar weightA = 1., scalar weightB = 1.)
  friend AAScalar convHullMinBounds(AAScalar &a, AAScalar &b, scalar weightA = 1., scalar weightB = 1.):
  Both return an AAScalar that represents a convex hull of $\widehat{a}$ and $\widehat{b}$ (see Section 4.3.3). The first one, convHull(...), blends between the two forms with an unknown value $\widehat{t} = [0, 1]$. The second simply constructs the minimal interval that encloses both $\widehat{a}$ and $\widehat{b}$. The new typical value is computed as the weighted sum of the typical values of both affine forms.

5.1.3 AAPoint

3D affine forms are implemented by this class. An AAPoint object is a vector of three AAScalars. It provides sum, difference, the dot product and the cross product of two AAPoints as well as the scalar product. The affine form for the length of a vector can be returned by length(). Most of the operations on AAPoint vectors are reduced to arithmetic operations on AAScalars.

A normalized AAPoint is generated by normalize() or normalize2(). The first function applies the standard formula $\frac{\widehat{\underline{n}}}{\|\widehat{\underline{n}}\|} = \frac{1}{\sqrt{\widehat{x}^2+\widehat{y}^2+\widehat{z}^2}} \cdot \widehat{\underline{n}}$, the second that is newly implemented uses the way that is described in Section 4.4.4. Like AAScalar this class is augmented by convHull(AAPoint &a, AAPoint &b, ...) and convHullMinBounds(AAPoint &a, AAPoint &b, ...) which determine a convex hull of two AAPoints. Note, that within convHull(...) all three components must be blended by the same affine form $\widehat{t}$.

5.1.4 AAShowPoint

In Appendix C it is explained how a visualization of a 3D affine form may be obtained. In order to keep the AAPoint class small this function is implemented in its own class because it needs additional data structures. The AAShowPoint class is a friend class to the AAScalar class to gain direct access to the coefficients of the affine forms. This class provides output in form of a VRML 2.0 file. The public functions are:

- AAShowPoint(const AAPoint &):
  Since this class should visualize 3D affine forms, an AAPoint must be given as parameter to the constructor.

- void openVRML(ofstream &):
  writes a VRML header on the already opened output file stream.

- void closeVRML(ofstream &):
  closes the VRML file.

- void outputVRML(ofstream &):
  A representation of the convex hull of the 3D affine form is written to the output stream as a set of facets.

5.1.5 AAColor

The AAColor class is used to handle tri-stimulus colors. One AAScalar object represents one of the three components (r,g,b). It is possible to add, subtract, multiply, divide and mix two AAColors which is done for each component. Again, even this class is extended by convHull (AAColor &a, AAColor &b, ...) and convHullMinBounds(AAColor &a, AAColor &b,...).

5.2 Vision

Lots of physically based rendering methods have been invented during the last decades which solve for approximations of Kajiya's rendering equation. They all need some sort of data structures representing the environment which includes the geometry of objects, their reflective properties, their emissive energy, the position of the camera, ... Also the ability to read in this data and to save or to render the results are part of these algorithms that reappear in others. To ease the reusage of these common parts, the Vision-system [17] has been developed at the computer graphics group at the University of Erlangen-Nürnberg. It builds a platform on which most of the contemporary known global illumination algorithms may be easily implemented. In order to achieve this the visualization process is divided into different steps that are supposed to be general enough that they do not conflict with any specific algorithm and allow to reuse parts of them within different implementations.

All quantities within the Vision framework possess a physical unit that ensures a homogeneous interface. Vision provides a program library written in C++ and a user interface based on Itcl/Tk. The object oriented approach makes it easy to derive from the given base classes to fit and to add a new algorithm without the need of reimplementation.

The Vision-architecture is arranged in the following independent modules that communicate across well defined interfaces:

Geometry: This module describes the geometrical properties of the scene's objects. Important within this radiosity algorithm are the position, the normals, the area, the size and the patch parameters.

Shader: The reflection properties (BRDF) and the emission for each object is handled by this module. It is separated from the geometry since several geometrical objects may share the same surface properties.

Scene-database: Since the geometry and the surface attributes are separated in Vision but one object possesses both there must be a mapping between the two. The entire scene description is stored in a directed acyclic graph (DAG). Inner nodes assign attributes like transformations, textures, illumination and so on to subordinated nodes whereas the leaves contain the actual object and geometrical description. The run through all objects, which means through all leaves, is implemented by a special class VisTraverser that returns the objects and their surface description.

Renderer: It determines those surface points depending on the viewing paramters for which illumination has to be calculated and initiates the computation. Ray-tracing or z-buffering may be used to choose those points.

Lighting: This module is responsible for the global illumination computation. At the beginning of each visualization process a lighting-object is enforced to perform the computation. Later, the object is asked for the resulting illumination for the given surface points. Every global illumination method derives a new lighting class.

View: Here all information needed for the entire visualization process is collected, as there is: the virtual camera with the viewpoint, the part of the scene graph that will be displayed, a renderer-object, a lighting-object and a picture-object that stores the results.

5.3 Implementation of the Affine Form Radiosity

5.3.1 General Changes in Vision

In order to enable the use of the affine arithmetic library within the Visison-system an AAScope must be instantiated before any other object of this library. This is ensured by creating one AAScope within the constructor of the VisWorld class that keeps track on all resources and data used during the entire process, which means I/O and memory management. The whole illumination computation is performed within one single scope.

Only during the solution process the affine arithmetic library is used. Pre- and post-processing is provided by the origin Vision-system which of course possesses its own classes for colors (VisColor) or points (VisPoint). Conversion of VisColor into and from AAColor is implemented by

- AAColor toAAColor(const VisColor&)
  that converts a VisColor into a AAColor,
- VisColor toVisColor(const AAColor&)
  that constructs a new VisColor of the typical values of the AAColor's components.

Both functions are added to the VisColor class. The VisPoint class is augmented by

- AAPoint toAAPoint(const VisPoint&)
  and
- VisPoint toVisPoint(const AAPoint&)
  that again works on the typical values.

The class VisTransform that implements all possible transformations on VisPoints had to be extended to suite AAPoints too.

5.3.2 Extension of the Geometry Subsystem

All kinds of geometrical objects in a scene are represented by subclasses of the VisGeoObject class. Within lots of algorithms these entire surfaces are recursively subdivided and geometrical information is needed also about the subdivided patches. Since there are many different requirements for subdivision of surfaces this concept is implemented separately by the VisSubdivision class whose objects may be subdivided generating a set of further VisSubdivision objects. The class VisGeoObject allows to obtain a VisSubdivision object for each geometric object, that supports subdivision. Each class of geometric primitives should subclass from

VisSubdivision to provide the class specific subdivision method. An algorithm does now query the VisSubdivision object for the geometrical properties of the patch such as area, size, convex hull, cone of normals, and so on.

In addition, the interface to the VisSubdivision class needs to permit queries for 3D affine forms that bound the normals and the spatial extent of the object as explained in Chapter 2 if the described radiosity algorithm should be applied. The function getAffinePoint(AAPoint *) answers the query for the bounds on the spatial extent and getAffineNormal(AAPoint *) returns an affine form that contains all normals. Both functions may be called several times for the same object during the solution process. It must be ensured that each time identical affine forms are returned by each call and that the normals and the spatial extent are generated by the same affine forms of surface parameters. Otherwise the underlying relationship between these quantities would be lost.

These conditions are fulfilled within the base class VisSubdivision: Once a 3D affine form for the normals or for the spatial extent is constructed, its value is cached. In addition, when one of the functions getAffineNormal(AAPoint *) or getAffinePoint(AAPoint *) is queried, both values are computed at the same time by a call to selfGetAAPointAndNormal(AAPoint *p, AAPoint *n) although just one value is returned. Subclasses of VisSubdivision must implement this virtual method and compute the affine forms in a class specific way. Corresponding subdivision classes exist for these classes of geometric primitives: VisRectParGeoObject, VisPlane, VisQuadric, VisPolygon, VisParallelogram and VisSphere which all are subclasses of the VisGeoObject class.

5.3.3 Implementation of a New Lighting Class

VisARLightOp

Like for each other global illumination algorithm that is integrated in the Vision architecture a new lighting class had to be derived for this affine form radiosity algorithm. The solution process of this algorithm is controlled by the VisARLightOp class that subclasses from VisLightOp. Stearing parameters are given to the constructor of the VisARLightOp object; they are explained in Table 5.1. Two methods define the lighting interface to the rest of the Vision system. Firstly, a call to prepareIllumination() starts the computation of a radiosity solution on the given scene. Afterwards, the result at specified surface points can be asked for by getIllumination(VisIllumReceiver *). The VisARLightOp class also provides other services for internal use.

- bool oracle(VisARObject *rec, VisARObject *sen, VisARLink *link, AAScalar cAvg, bool totvis):
  implements the oracle that makes the error driven refinement decision. The approximated form factor interval is retrieved from the link and then multiplied by the radiosity of the sender that is averaged over the three components. If only partial visibility is encountered between sender and receiver the user defined error threshold

Parameter	Standard value / Type	Description
shooting	false / bool	selects whether the solution is calculated using gathering or shooting.
oracle	<u>AFFINE</u>, FORMFACTOR, SAMPLED	uses an approximation of the form factor within the oracle that is computed with affine forms, or by sampling together with BF- or error driven refinement.
norm	<u>L1</u>, L2, Linfinity	the applied error norm.
error	0.01 / double	user defined error threshold
partvisfac	10 / double	partial visibility factor
min area	0 / double	minimum area that allows subdivision
max iter	5 / int	number of iterations
vis factor	15 / double	factor used to determine number of visibility tests
vis min	4 / int	minimum number of vis. tests
vis max	20 / int	maximum number of vis. tests
double-sided	false / bool	allows double-sided surfaces
closed	false / bool	The scene is closed. The form factor of each patch must sum to 1 and can be corrected.
output	<u>TYPICAL</u>, MINIMUM, MAXIMUM, ERROR	chooses the output value
error of	<u>APPROX</u>, CENTER, UNKNOWN	selects the typical value
use interval	false / bool	propagates intervals to the next iteration

Table 5.1: Parameters of VisARLightOp objects

is lowered by a given factor. The refinement will stop if the area of the object selected to be subdivided falls below some minimum value.

- void makeVisTests(bool, int, VisARObject *, VisARLink *, VisARObject *):
  performs visibility tests. It is called twice for each pair of objects, first to test for partial visibility and then to gain a quantitative result when energy is exchanged across the current link. The visibility is stored in each link. Sampling or affine forms may be used.

- VisDouble ffcomp(VisARObject *rec, VisARObject *sen):
  returns a sampled form factor between the sender and receiver.

VisARBuildTraverser

As described above, the leaves of the scene graph representing the geometrical objects have to be accessed through a special traverser. The VisARBuildTraverser class, a subclass of VisTraverser, encapsulates the scene graph and builds the data base for this radiosity algorithm. It creates a VisARRootPatch object for each scene patch and a VisARCluster object for each cluster.

VisARLink

The connection between two objects is represented by a VisARLink. It stores all data about the correlation of these objects that will stay constant during the entire solution process. This way links may be reused in later iteration steps. The invariant quantities are the form factor, the incomplete form factor of clusters, the results of the visibility tests and the decision whether the sender, the receiver or both objects should better be subdivided when refinement is enforced. It ensures the recorded form factor to lie within appropriate bounds, that is the interval $[0, 1]$.

VisARObject

This class is a virtual base class that collects all relevant data of an object in the scene; that are both, geometrical and reflective properties, as well as subdivided child patches. It also implements a lot of the actual hierarchical radiosity algorithm. Important methods are:

- VisConvexHull *getConvexHull()
  VisDouble getSize()
  VisDouble getArea()
  bool getAffineNormal(AAPoint *)
  bool getAffinePoint(AAPoint *):
  supply all necessary geometrical information. These methods must be implemented by the subclasses.

- void subdivide()
  list<VisARObject *> &getChildren():
  each VisARObject can be subdivided, and provides a list of its children.

- bool isConvex():
  denotes if the object is convex.

- bool boundProjArea(const AAScalar &cosAlpha, AAScalar *):
  calculates upper and lower bounds on the projected area of the sender seen from the receiver. The direction is indirectly included in the term cosAlpha; this way recomputation is avoided.

- bool boundIntVis(const AAScalar &cosAlpha, AAScalar *intvis):
  bounds the internal visibility.

- AAColor &getEmission():
  yields the emission.

- bool boundRefl(AAColor *)
  bool boundRefl(const AAPoint &dir, AAColor *):
  bounds the reflection of the object. If dir is given all subpatches that do not contribute to the reflection in this direction are ignored.

- bool boundRad(AAColor *)
  bool boundRad(const AAPoint &dir, AAColor *):
  returns the radiance of the object with respect to the given direction.

- AAColor &getOutRad()
  AAColor &getInRad()
  AAColor &getInRadSum():
  within a hierarchical radiosity algorithm each object must keep track on its unshot radiosity, its directly gathered radiosity and the totally gathered radiosity through all iteration steps.

- void addInRad(const AAPoint &,const AAScalar &ff, const AAColor &rad):
  adds the irradiance from a given direction, that is the sender's radiosity times the form factor. The sum of gathered form factors also is updated. It is used to determine a typical form factor for self-interaction or correction.

- void smartAddInRad(AAPoint &dir, AAScalar &ffold, AColor &rad):
  this method is used when radiosity is gathered by a cluster. The form factor is supposed to not include the cosine term of the receiver. The radiosity is right away pushed down to the children where the correct form factor is computed using the children's cosine term. No radiosity is added directly to the irradiance of the cluster.

- void pushPull(bool em, AAColor = AAColorBlack):
  performs the push/pull operation as described. The outgoing radiosity is updated after multiplication times the reflectance.

- void setLink(VisARLink *link, VisARObject *sender, ...):
  this method computes the bounds as well as the typical value of the unoccluded form

factor between the object and the sender using the formula of Section 4.4. The link records the result. If the receiver is a cluster, the incomplete form factor that lacks the receiver's cosine is stored too.

- void setSelfLink(VisARLink *link, VisDouble ffSumPushed):
  bounds the form factor for self-interaction and stores the result with the self-link.

- void propagate(VisARLink *, VisARObject *sender, bool = false)
  void propagate(VisARLink *, VisDouble = 0.):
  transports the unshot radiosity of the sender to the current object. If no sender is given self-interaction has to be considered. This is done the following way.

  1. The form factor is approximated by setLink(...) or setSelfLink(...).
  2. The initial qualitative visibility test is performed. Of course this step is neglected if the result is already given by the parent object.
  3. The oracle is consulted.
  4. If further refinement is necessary the link and the sender or the receiver are subdivided and the propagate(...) method is called recursively for the generated children.
  5. Else, the concrete visibility between the objects is sampled and the transported radiosity is added to the irradiance of the receiver.

During one iteration step the VisARLightOp object sequentially calls the pushPull(...) and the propagate(...) method of the VisARObject that represents the entire scene with one initial link as parameter. Since the self-form factor of the complete scene normally cannot be approximated with sufficient quality, the scene is subdivided and links are established between the scene's children, and so on.

Several subclasses of the VisARObject class exist that override some of the above member functions. The class hierarchy is depicted in Figure 5.1. The subclasses also make available some helper functions for the visibility tests.

VisARPatch

This class handles subdivided patches in the scene. It stores a pointer to its entire root patch. All information about the shader is accessed through the root patch whereas the geometrical data is directly obtained from the corresponding VisSubdivision object. If one VisARPatch object is subdivided the same is done to the VisSubdivision object.

The smartAddInRad(...) method that ensures correct gathering within clusters computes the receiver's cosine of this patch, completes the unfinished form factor and adds the incoming radiosity to its irradiance.

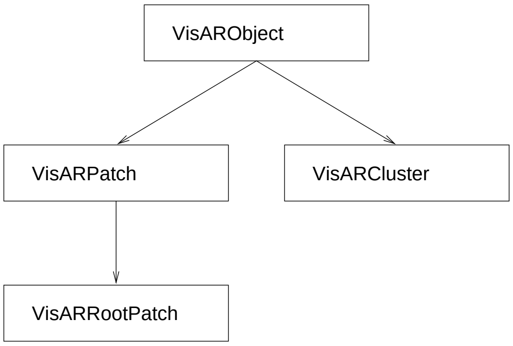


Figure 5.1: Class hierarchy of the VisARObject class.

VisARRootPatch

The VisARRootPatch class is a subclass of VisARPatch. Each initial scene surface results in a VisARRootPatch object. As the shader properties are defined across the entire surface they are the same for each child patch and thus may be accessed through the root patch. The same is true for the isConvex() predicate. The reflectance and the emission of the VisSubdivision object are obtained from the shader subsystem by calls to getMeanReflectance() and getMeanEmission().

VisARCluster

For clusters no corresponding VisSubdivision object exists. All information must be obtained and collected from the cluster's children which of course are already accessible before a cluster is subdivided since the cluster hierarchy is constructed bottom up. The single quantities are computed as described below:

- The area of the cluster equals the sum of the children's area.
- The size is determined by the length of the diagonal of the bounding box.
- The two bounding 3D affine forms for the spatial extent and the normals must conservatively bound all of the children's affine forms (see Section 4.3.3).
- Minimum and maximum value of the cluster's reflectance is given by the minimum and maximum reflectance among all children. The typical value like the emission is computed as the area-weighted sum of the children's values.
- Clusters will only be considered convex in the special case of a box.

In order to respect the anisotropic character of clusters the ideas on the clusters' cosine terms of Section 4.4.5 have been implemented. The addInRad(...) like the smartAddInRad(...) method of clusters only needs to call the smartAddInRad(...) methods of the children. Besides this, the exitant radiance with respect to a given direction is bounded by boundRad(...) neglecting all children that are actually back-facing.

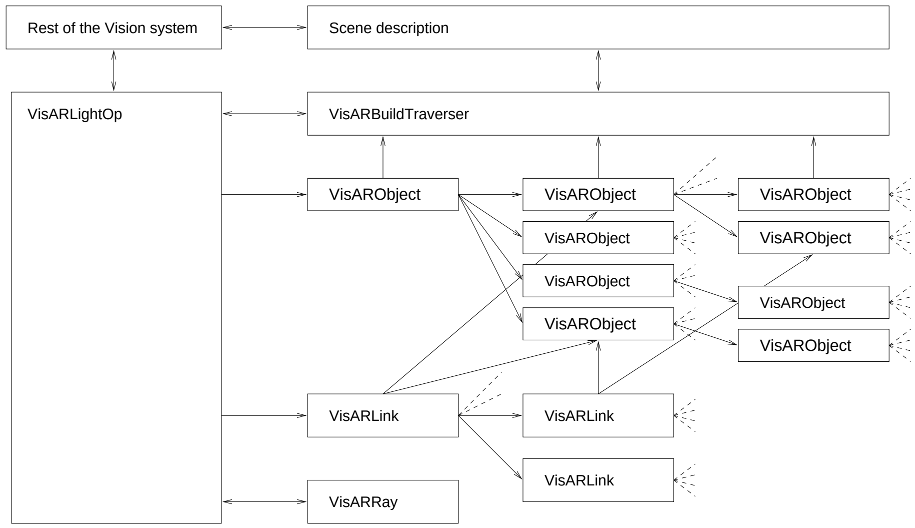


Figure 5.2: The most important classes of the affine form radiosity implementation within the Vision system.

VisARRay

This class encapsulates the interface of the VisRay class in such a way that rays may be set up between two VisARObjects. It is applied to sample the visibility.

The most important classes of this implementation and their relations are depicted in Figure 5.2.

Chapter 6

Results

6.1 Comparison to Bounded Radiosity

After the influence of different ways of computing the single terms of Equation 4.3 has been discussed and examined in Chapter 4, the overall behavior of the implemented algorithm is investigated in this section. The affine arithmetic approach has been compared against the bounded radiosity algorithm using six scenes with varying characteristics:

- **Scene A** (Figure A.1): Two parallel unit squares one unit distance apart; one acts as the sender, the other as receiver. The analytic form factor for this scene is approximately 0.1998 (see [2]).
- **Scene B** (Figure A.2): Two perpendicular unit squares that share a common boundary, resulting in a singularity. Its analytic form factor is 0.200043 (see [2]).
- **Scene C** (Figure A.3): Four chairs grouped around a table at the center of a closed room. The scene is lit by an area light source.
- **Scene D** (Figure A.4): An office filled with several desks and shelves. The clusters of this scene tend to be more complex than those in Scene C.
- **Scene E** (Figure A.5): A columned hall with a sculpture in its center. The columns and the sculpture consists of cylinders and cones.
- **Scene F** (no figure): Like Scene C, but the emission and the reflectivity of each patch are constrained by the equation $\rho = 1 - 2E$. The resulting radiosity will be constant over the whole environment and its value is $B = E + \rho B = 1/2$, no matter how complex the scene is .

The results of the affine form radiosity are computed using the following choices: The typical value of the form factor is determined by sampling, clusters are bounded by their bounding boxes, the min-range approximation is used for $x \to \frac{1}{x}$ and $x \to x^2$ and visibility tests are performed by ray-casting. As can be seen in Table 6.1, the number of links that are produced by calculating the form factor using affine forms is considerably smaller than

Scene	Norm	Iterations	Oracle Threshold	Affine		Bounded	
				Links	Time	Links	Time
A	L_2	1	0.0001	12425	5s	36713	3.6s
B	L_2	1	0.0001	619985	175s	640017	61s
C	L_2	5	0.001	622561	370s	1262816	356s
D	L_1	5	0.01	430850	864s	910811	1228s
E	L_1	5	0.07	557489	516s	623405	583s
F	L_2	18	0.001	1772521	32 min.	3859856	24 min.

Table 6.1: Links generated using affine form radiosity and bounded radiosity.

the number of links generated by the bounded radiosity algorithm applying the same error threshold. This shows that affine arithmetic actually results in tighter bounds on the form factor than standard interval arithmetic does. But unfortunately, the more complex computation of affine forms increases the consumed time in such way that the advantage of precise bounds is more than compensated.

Another drawback is the amount of storage that is occupied by each link. To store two affine forms in each link, one for the complete and one for the cluster's form factor, of course blows up the size of an `AALink` object compared to a representation using one simple interval.

Besides the number of created links the quality of the produced subdivision is of interest too. The error of each link is not necessarily a direct measure of the quality of the produced solution, although the results will be more precise if the oracle threshold is lowered. Therefore, the computed approximation for Scene A, B and F is compared to the analytical solution with respect to the number of generated links. In Figures 6.1 (Scene A) and 6.2 (Scene B) the determined form factor is plotted for different numbers of links. Figure 6.3 depicts the average value of the computed radiosity for Scene F and Figure 6.4 shows the mean distance to these average values which reflects the homogeneity of the computed solution.

Bounded radiosity seems to approximate the analytical solution slightly better for a huge amount of links whereas the computation with affine forms does better for small numbers. BF-refinement sometimes even produces more precise results although it underestimates the radiosity in Scene F.

6.2 Conclusion and Future Perspectives

In this thesis the applicability of affine arithmetic to a hierarchical radiosity algorithm with error driven refinement has been investigated. The approach succeeds to calculate tight conservative bounds on the form factor. But it exhibits the drawback of increased computational costs in both time and storage. As explained in Chapter 4, sometimes tighter bounds are obtained if a computed affine form for a single term is replaced by an interval when upper or

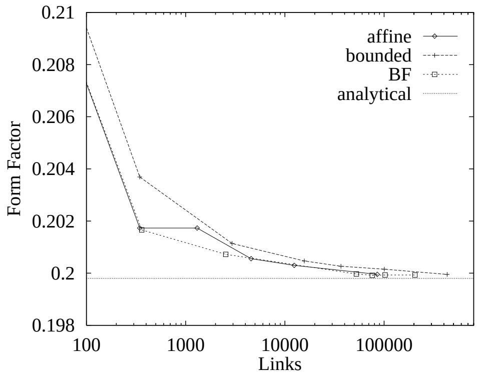


Figure 6.1: Computed form factor of Scene A

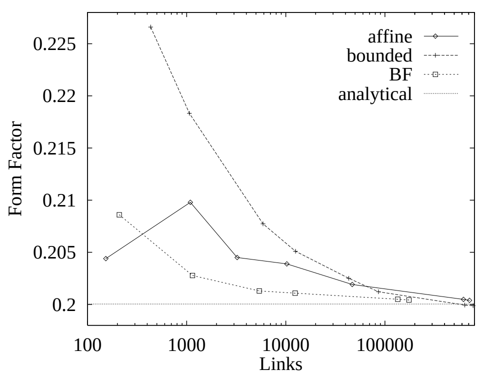


Figure 6.2: Computed form factor of Scene B

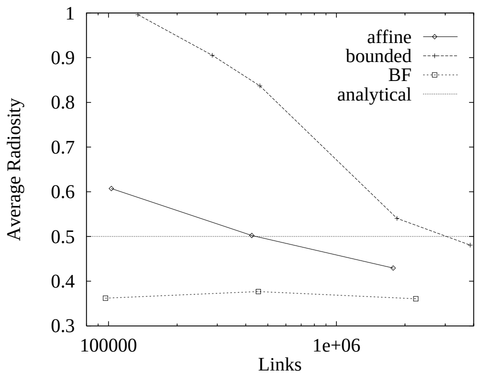


Figure 6.3: Average value of computed radiosity of Scene F

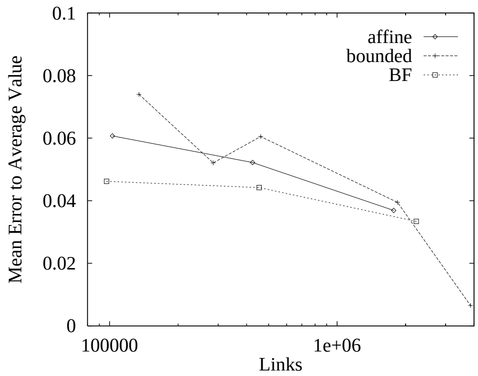


Figure 6.4: Mean error to the average of the computed radiosity values of Scene F

lower bound are constrained due to other considerations. Thus, it was not possible to improve some parts of the algorithm by the use of affine arithmetic.

The geometrical properties of each involved object, no matter if it is a curved or a planar patch or a cluster, has been represented by 3D affine forms that allow treating solid objects like they were single points within the form factor computation. This is a fascinating idea but also has its limits. An appropriate bounding affine form has to be found for the spatial extent of each kind of object, which is easy for rectangular parameterized surfaces because the 3D affine form can be derived directly from the surface description. Even for triangles and convex polygons an optimal representation with affine forms can be found, but for clusters or other objects of arbitrary shape this is not easily possible. For clusters the bounding box is used instead to represent the spatial extent.

3D affine forms also are used to bound normals and directions between objects. In particular, affine form normals of clusters and the bounds on directions between neighboring objects may contain vectors of zero length, and normalization cannot be performed in the standard way. To overcome this problem a method has been developed which normalizes one component of the vector at the time avoiding singularities due to zero length.

Bounds on the patch-to-patch form factor are calculated by use of Equation 4.3 which is evaluated by affine arithmetic. Several quantities contributes to the result:

- The cosine terms of sender and receiver are determined by the dot product of the 3D affine forms bounding the normals and the corresponding direction.
- The projected area of the sender is computed by the area times the sender's cosine. Although the upper bound may be lowered by the maximum of the projected area of the 3D affine form, the overall result is hardly improved.
- To calculate the denominator $\frac{1}{\|\hat{\underline{d}}\|^2}$ the min-range approximation has to be used for the square in order to avoid unnecessary singularities.

In the case of clusters, even an incomplete form factor is computed that misses the cluster's cosine term which will be replaced by a child's cosine if the cluster gathers radiosity.

Besides the affine form bounding the form factor, a typical value has to be determined that approximates the actual form factor. One of three values may be chosen: the center of the affine form, the typical value that is computed simultaneously with the bounds or a sampled approximation. The last alternative achieves the best visual results but is even the most expensive.

Patches are refined if the maximum error resulting when the form factor across one link is approximated by a given typical value still lies above the oracle threshold. Which object will be refined is determined by a criterion that involves the area and the curvature of both objects. Neglecting the curvature would generate more links in complex scenes with clusters or curved surfaces.

When radiosity has been gathered by some patch it must be propagated down and up the patch hierarchy by the push/pull operation. The radiosity of a parent patch is bounded by the minimum and maximum value of the radiosity of its children. By this step all information about error variables is lost, the radiosity is reduced to a simple interval. Thus, affine arithmetic computation may stop after the form factor has been determined.

Finally, we have investigated the use of affine arithmetic for visibility tests, unfortunately with little success: the product of two affine forms always contains a new independent source of error that prohibits precise representation of the rays between two objects.

The implementation of course could be improved further. The computation may be accelerated if those parts of the algorithm that actually handle intervals are implemented using a standard interval arithmetic library that only keeps track of the upper and lower bounds and the typical value. This way the benefits of both approaches affine and interval arithmetic can be combined. The need of storage may be significantly reduced if only those links of the upper levels of the hierarchy are stored for reuse in the next iteration step.

In addition, each term of Equation 4.3 contributing to the resulting bounds on the form factor may be investigated further. Perhaps other ways to compute them can be found which produce even tighter bounds. A better yet not optimal representation of the normals and the spatial extent of clusters than the bounding box will further tighten these bounds when a cluster is involved.

Although affine arithmetic is not suitable for visibility test, 3D affine forms allow a simple intersection test (see Section 4.7) and are far more flexible than spheres or bounding boxes to bound objects of arbitrary shape. This may be used within other computer graphics contexts, i.e. in VR applications, to determine intersections of moving objects.

Appendix A

Pictures

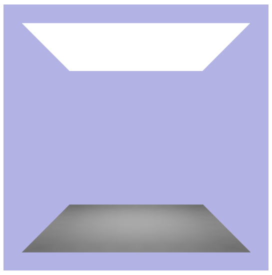


Figure A.1: Scene A

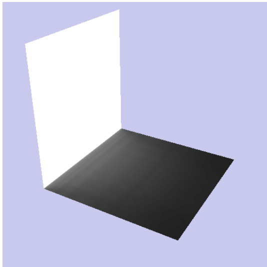


Figure A.2: Scene B

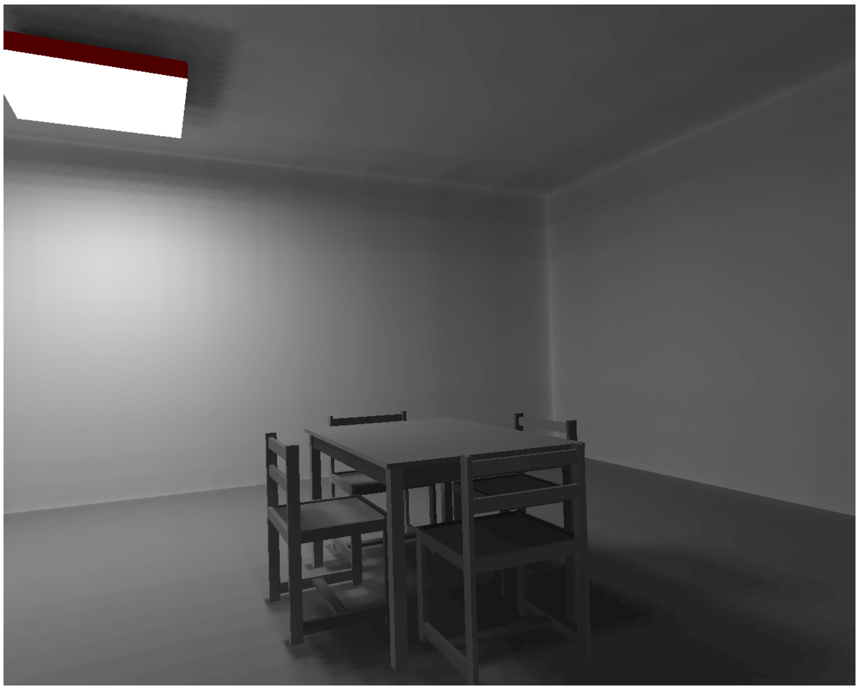


Figure A.3: Scene C

Figure A.4: Scene D

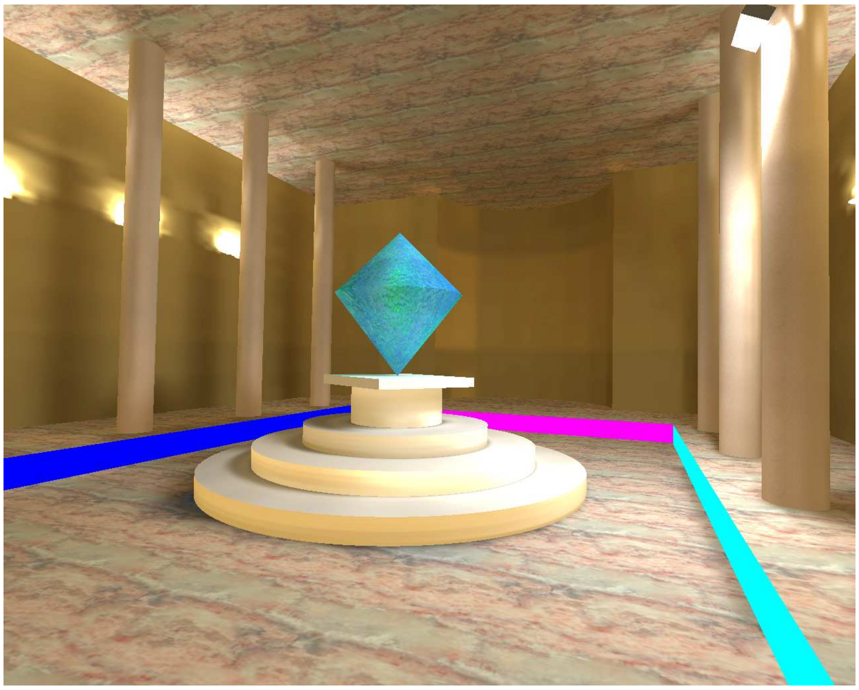


Figure A.5: Scene E

Appendix B

Area computation of 2D affine forms

The range of two-dimensional affine forms is bounded by a center-symmetric polygon with edges that are parallel in pairs. The area of this range is somewhat smaller than the area of its bounding box if the two components $\widehat{x}$ and $\widehat{y}$ share some error variables. Error variables that are found in only one of the components do not decrease the area. The pairs of coefficients (x_k, y_k) of all other variables ε_k have to be sorted by their slope x_k/y_k in decreasing order. The area of the bounding box is first reduced by the sum of the area of all triangles t_k that is

$$A_t = \sum_{i=1}^{n} 4x_i y_i \tag{B.1}$$

and then by the area of the rectangles r_k. Those rectangles must be treated separately for $x_k/y_k > 0$ and $x_k/y_k < 0$. Let there be p error variables of positive and q of negative slope. The summed area of the rectangles is given by

$$A_r = 2\left(\sum_{i=2}^{p} 4y_i \left(\sum_{j=1}^{i-1} x_j\right)\right) + 2\left(\sum_{i=p+2}^{p+q} 4x_i \left(\sum_{j=p+1}^{i-1} y_j\right)\right). \tag{B.2}$$

The total area of the 2D affine form now results in

$$A_{\max} = A_{BBox} - A_t - A_r. \tag{B.3}$$

This method is depicted in Figure B.1. Note, that this is the upper bound on the area that is covered by the represented quantity. For the lower bound without further knowledge no better value can be found than zero. Thus, $\widehat{A} = [0, A_{\max}]$. This method is not an affine approximation since all error variables are lost.

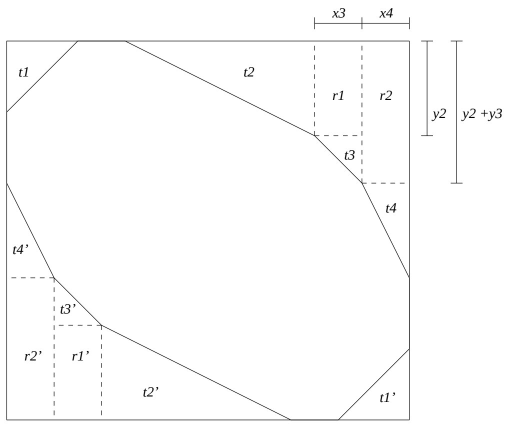


Figure B.1: Computing the area of a 2D affine form

Appendix C

Visualization of 3D affine forms

The surface of the range of a 3D affine form can become arbitrary complex depending on the number of error variables. Assume, a 3D affine form $\underline{\widehat{a}}$ is represented by

$$\begin{aligned}\underline{\widehat{a}} &= \begin{pmatrix} \widehat{x} \\ \widehat{y} \\ \widehat{z} \end{pmatrix} = \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} + \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} \varepsilon_1 + \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} \varepsilon_2 + \cdots + \begin{pmatrix} x_n \\ y_n \\ z_n \end{pmatrix} \varepsilon_n . \\ &= \underline{x}_0 + \underline{x}_1\varepsilon_1 + \underline{x}_2\varepsilon_2 + \cdots + \underline{x}_n\varepsilon_n \end{aligned} \tag{C.1}$$

Starting with the center point $\underline{x}_0 = (x_0, y_0, z_0)^T$ the solid body of the range may be successively constructed by sweeping $\underline{x}_0$ along $\underline{x}_1$ and $-\underline{x}_1$. The resulting line then is swept along $\underline{x}_2$ and $-\underline{x}_2$, and so on (see Figure C.1).

In order to yield a usable surface description little more work has to be done since successive sweeping along n vectors produces 2^n points, although most of them don't lie on the boundary. Assuming that no two vectors are equal, the algorithm works as follows and maintains a point, an edge and a patch list. Initially, the point list only contains the center point, the edge list and the patch list are empty. For each vector $\underline{x}_i$ the point list is doubled representing the swept points in both directions. Every point that will lie inside the new object is erased. If both points survive a new edge is introduced. Each of the initial edges results in two new ones if the corresponding nodes are still valid, else only one new edge is constructed. Two related new edges span a new patch. The patches are updated the same way like edges. Only if all swept nodes of one direction still lie on the boundary that patch remains on the list. Last, if $\underline{x}_i$ sweeps the points coplanar to one patch, this patch will change its shape and two new points must be added. These points are connected by edges that are not yet part of two patches, and are thus easy to find. The remaining points of both point lists are collected into a single one and the next vector can be processed.

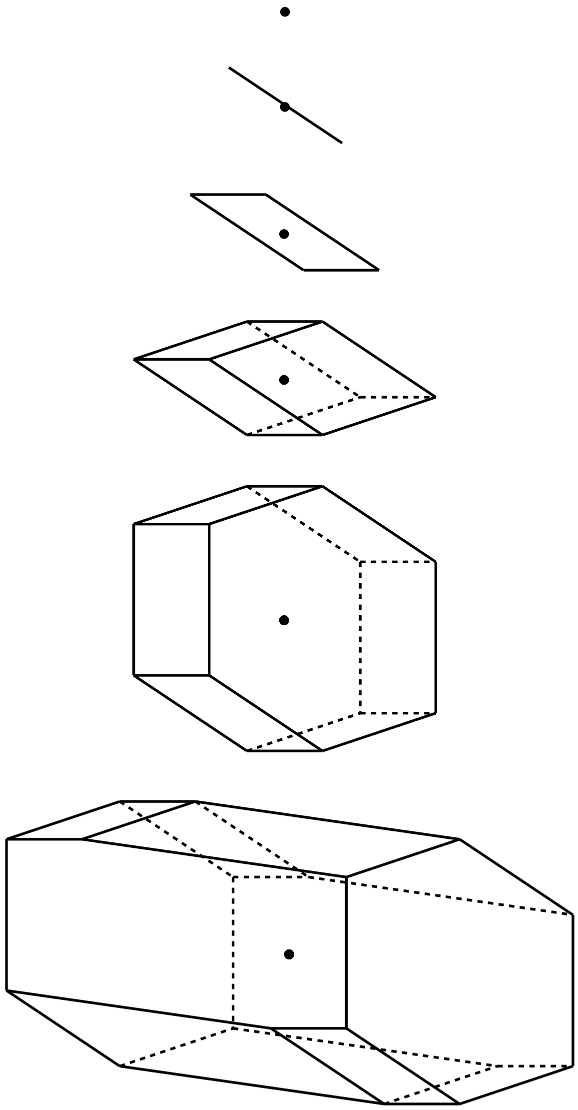


Figure C.1: Constructing the range of a 3D affine form

Bibliography

[1] James Arvo, Kenneth Torrance, and Brian Smits. A framework for the analysis of error in global illumination algorithms. In *Computer Graphics (SIGGRAPH '94 Proceedings)*, pages 75–84, July 1994.

[2] Michael F. Cohen and John R. Wallace. *Radiosity and Realistic Image Synthesis*. Academic Press, Boston, 1993.

[3] João L. D. Comba and Jorge Stolfi. Affine arithmetic and its applications to computer graphics. In *Anais do VII Sibgrapi*, pages 9–18, 1993.

[4] Luiz Henrique Figueiredo. Surface intersection using affine arithmetic. In Richard Bartels, editor, *Graphics Interface '96*, pages 168–175, 1996.

[5] Luiz Henrique Figueiredo and Jorge Stolfi. Adaptive enumeration of implicit surfacess with affine arithmetic. In *Computer Graphics Forum*, 15(5):287-296, 1996.

[6] S. Gibson and R. J. Hubbold. Efficient hierarchical refinement and clustering for radiosity in complex environments. In *Computer Graphics Forum*, 15(5):297-310, December 1996.

[7] C. Goral, K. Torrance, D. Greenberg, and B. Bataille. Modelling the interaction of light between diffuse surfaces. In *Computer Graphics (SIGGRAPH '84 Proceedings)*, 18(3):211-222, July 1984.

[8] Pat Hanrahan and D. Salzmann. A rapid hierarchical radiosity algorithm for unoccluded environments. In *Eurographics Workshop on Photosimulation, Realism and Physics in Computer Graphics*, 1989.

[9] Paul Heckbert. Discontinuity mesching for radiosity. In *Proceedings of the Third Eurographics Workshop on Rendering* (Bristol, UK), pages 203–226, May 1992.

[10] Wolfgang Heidrich and Hans-Peter Seidel. Ray-tracing procedural displacement shaders. In *Graphics Interface '98*, pages 8–16, 1998.

[11] Wolfgang Heidrich, Philipp Slusallek, and Hans-Peter Seidel. Sampling procedural shaders using affine arithmetic. *ACM Transactions on Graphics*, 1998.

[12] James T. Kajiya. The rendering equation. In *Computer Graphics (SIGGRAPH '86 Proceedings)*, 20(4):143-150, August 1986.

[13] Dani Lichinski, Brian Smits, and Donald P. Greenberg. Bounds and error estimates for radiosity. In Andrew Glassner, editor, *Computer Graphics (SIGGRAPH '94 Proceedings)*, pages 67–74, 1994.

[14] R. E. Moore. *Interval Analysis.* Prentice-Hall, Englewood Cliffs, NJ, 1966.

[15] François X. Sillion. A unified hierarchical algorithm for global illumination with scattering volumes and object clusters. In *IEEE Transactions on Visualization and Computer Graphics*, 12(3):261-272, September 1993.

[16] François X. Sillion and Claude Puech. *Radiosity and Global Illumination.* Morgan Kaufmann Publishers, San Francisco, 1994.

[17] Philipp Slusallek. *Vision - An Architecture for Physically Based Rendering.* Phd thesis, University of Erlangen, IMMD 9, Computer Graphics Group, April 1995.

[18] Brian Smits, James Arvo, and Donald Greenberg. A clustering algorithm for radiosity in complex environments. In *Computer Graphics (SIGGRAPH '94 Proceedings)*, pages 435–442, July 1994.

[19] John M. Snyder. Interval analysis for computer graphics. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 121–130, July 1992.

[20] Marc Stamminger, Philipp Slusallek, and Hans-Peter Seidel. Bounded radiosity - illumination on general surfaces and clusters. In *Computer Graphics Forum (EUROGRAPHICS '97 Proceedings)*, 16(3), September 1997.

[21] Marc Stamminger, Philipp Slusallek, and Hans-Peter Seidel. Bounded clustering - finding good bounds on clustered light transport. In *Proc. Pacific Graphics '98.* IEEE Computer Society Press, 1998.

[22] Jorge Stolfi and Luiz Henrique de Figueiredo. Self-validated numerical methods and applications. Technical report, Instituto de Computacao, UNICAMP, May 1997.

[23] Seth Teller and Pat Hanrahan. Global visibility algorithms for illumination computations. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, pages 239–246, August 1993.

[24] Roy Troutman and Nelson L. Max. Radiosity algorithms using higher order finite element methods. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, pages 209–212, August 1993.

[25] Alan Watt and Mark Watt. *Advanced Animation and Rendering Techniques - Theory and Practise.* Addison-Wesley, Reading, Massachusetts, 1992.

[26] Harold R. Zatz. A higher order solution method for global illumination. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, pages 213–220, August 1993.

Erklärung

Ich versichere, daß ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und daß die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Ich bin damit einverstanden, daß die Arbeit veröffentlicht wird und daß in wissenschaftlichen Veröffentlichungen auf sie Bezug genommen wird.

Der Universität Erlangen-Nürnberg, vertreten durch den Lehrstuhl für Graphische Datenverarbeitung, wird ein (nicht ausschließliches) Nutzungsrecht an dieser Arbeit sowie an den im Zusammenhang mit ihr erstellten Programmen eingeräumt.

Erlangen, den 23. September 1998

(Hendrik Lensch)