A Hardware Algorithm for Fast Realistic Image Synthesis

A. C. Yilmaz, S. Hagestein, E. Deprettere and P. Dewilde

A VLSI oriented algorithm, for the implementation of a generalized two-pass radiosity method is presented. The method allows any reflection behavior, varying from purely diffuse to perfect mirroring. Moreover, objects may be defined in terms of curved (Bezier) surfaces. All computations in the pre- and postprocess are similar and ray-tracing based, consequently a single architecture can be devised for both passes. This architecture, when built on ray-rotating and ray-tracing pipelined processors such as Cordics, results in a very high throughput VLSI implementation of the proposed generalized two-pass procedure.

1. Introduction

In recent years, new methods for realistic image rendering have been presented. Ray-tracing and radiosity methods appear to be most suitable for realistic image synthesis. Ray-tracing methods are conceptually simple and especially suited to model transparency and near mirroring reflections. The radiosity method, based on energy transport and conservation principles, models correctly the diffuse interreflection within an environment. It has been successfully applied in thermal heat transfer simulations [18,19] and was reintroduced in computer graphics by Goral et al. [9]. The method is computationally intensive since it requires the computation of the entries - the so-called form-factors - in a large matrix and the iterative solution of a set of linear equations involving this matrix. The computation of form-factors can be simplified by projecting the patch onto a hemicube [5], though this leads to an inhomogeneous sampling of the environment above the cube. We will use the original hemisphere instead of the hemicube. The accuracy of the diffuse intensity has been improved by an adaptive patch subdivision method [4] in which the rendering of shadows and penumbras are accounted for. The implementation proposed in [4] uses a hierarchical data structure requiring adjacency relationship within the environment. An alternative approach will be presented in this paper. A first attempt to extend the method to include non-diffuse surfaces was proposed by Immel et al. in [11]. Since this method computes a complete view-independent intensity distribution over all patches, an unacceptable high cube resolution is required.

A practical method to deal with complex environments was introduced by Wallace et al. in [20]. They proposed a two-pass approach consisting of a view-independent preprocess, based on the radiosity method, and a view-dependent postprocess, based on ray-tracing. In the preprocess, form-factors are computed by using a hemicube and a projection technique. In [20], so called "extra form-factors" representing the extra amount of reflected energy via perfect mirrors are taken into account. In [15] the restriction to perfect mirrors was removed by using a procedural iteration to approximate differently defined extra form-factors. Although this approach can reduce the processing time as compared to the approach presented by Immel [11], the following drawbacks can not be ignored. Firstly, in case of high intensity gradients, a lot of memory is needed to store the visibility information for each non-diffuse patch. Secondly, as with [4], a very high hemicube resolution is required to render the directional intensity gradient accurately.

In this paper we present a novel two-pass procedure for the visualizing of complex environments. The procedure is a modification to the approach in [20], in the sense that it is both generalized and specialized. It is more general, not only because it allows any reflection behavior, from purely diffuse to perfect mirroring, but also because it accepts curved (Bezier) surface descriptions. The procedure is specialized in that it is optimized in terms of hardware, in particular high speed ASIC implementation.

The first pass (*preprocess*): In this pass we compute form-factors by using a hemisphere and ray-tracing. A set of rays is used to define a reference hemisphere. Each hemisphere ray refers to a precomputed delta form-factor specifying the incoming energy within a small solid angle. This energy arrives from a diffuse patch, directly or indirectly via specular reflections. For each diffuse patch the rays are rotated, by using Cordic arithmetic, and cast. If the ray intersects a diffuse patch directly, the corresponding delta form-factor is added to the form-factor between the two involving patches. If the ray intersects the same diffuse patch indirectly, via specular reflections, a weighted delta form-factor is added to the form-factor. So, we compute correct intensities on diffuse patches by taking multiple specular reflections into account, but without computing specular patch intensities themselves. The size of the form-factor matrix is independent of the number of specular patches. The use of a hemisphere makes an efficient storage of the frustum (or the cone) weight-factors possible, so that reflections depending on a varying solid angle and a reflection power (see section 3.3) can be easily modeled. The storage problem of multiple reflection and transmission within varying frustums has been overcome by applying a backtracking procedure. All intersections are computed by using a subdivision and a bounding box algorithm. By using a bounding box algorithm, patches are no longer restricted to planar polygons but Bezier forms can be processed as well. Once the form-factor matrix is constructed the diffuse patch intensities are obtained by applying a Gauss-Seidel iterative matrix equation solver [3].

Instead of the above mentioned adaptive patch subdivision method we apply an "adaptive patch sampling" method, which avoids the need to using a hierarchical data structure requiring the adjacency information. The sample points on patches are controlled by parametric (u,v) coordinates.

The second pass (postprocess): In this pass we cast rays from a given eye position through the screen pixels and compute intersections with the corresponding patches. If the intersected patch is diffuse then the diffuse intensity component of the screen pixel is obtained by bi-linear interpolation using intensity values computed in the first pass. If the patch is specular then the specular intensity component is computed by shooting rays confined to a frustum.

It should be noted that shooting rays from a hemisphere in the preprocess similar to shooting rays through the displaying screen in the postprocess and the computation of extra light on diffuse patches, caused by multiple specular reflections, in the preprocess is similar to the specular intensity computation in the postprocess. Moreover, because both passes of the algorithm are ray-tracing based, a single architecture is naturally used to implement both passes. Since all rays computations are both similar and independent of each other, the whole procedure is optimally tailored towards a very fast SIMD/wavefront-type architecture.

The outline of this paper is as follows. In section 2 we give a concise review of the radiosity method. In section 3 we will show how light energy transfer can be modeled by defining, what we called, direct and indirect form-factors. Also in this section we will show how these form-factors can be computed by approximating patch to hemisphere projections. Adaptive patch sampling is described in section 4. In section 5, attention will be paid to the postprocess, including the view dependent specular intensity computation stage and the rendering stage. In section 6, a preliminary conceptual hardware version is given. In section 7 it is shown how Bezier patches are processed. Finally some statistics and software simulation results are given in section 8.

2. The Illumination Equation

Kajiya [12] and Immel [11] have shown that the intensity of light reflected by a surface obeys the following illumination equation:

$$I_{out}(\theta_{out}) = E(\theta_{out}) + \int_{\Omega} \rho''(\theta_{out},\theta_{in})\, I_{in}(\theta_{in}) \cos\theta \, d\omega, \qquad (2.1)$$

where

θ_{in} = the incoming direction, described by polar angle θ and azimuthal angle ϕ.
θ = the angle between the incoming direction and the patch normal.
$I_{out}(\theta_{out})$ = the intensity in outgoing direction θ_{out}.
$E(\theta_{out})$ = the outgoing intensity in direction θ_{out} due to emission.
$\rho''(\theta_{out},\theta_{in})$ = bidirectional reflectance/transmittance coefficient.
$I_{in}(\theta_{in})$ = the incoming intensity in direction θ_{in}.
Ω = the sphere of incoming directions
$d\omega$ = the solid angle through which I_{in} arrives.

In a number of papers [16,7,10] it was shown that with very little loss of accuracy, the bidirectional reflection coefficient can be approximated as the sum of a view-independent portion $k_d\rho_d$ and a view-dependent portion $k_S\rho_S(\theta_{out},\theta_{in})$. Further separation of both portions into a reflected and transmitted part yields:

$$\rho''(\theta_{out},\theta_{in}) = k_d\rho_d + k_y\rho_y + k_r\rho_r(\theta_{out},\theta_{in}) + k_t\rho_t(\theta_{out},\theta_{in}), \qquad (2.2)$$

where

k_d = diffuse fraction of reflected light energy
k_y = diffuse fraction of transmitted light energy
k_r = specular fraction of reflected light energy
k_t = specular fraction of transmitted light energy,

with $k_d + k_y + k_r + k_t \leq 1$.

Reflection models for surfaces have been described by Phong [16], Cook et al. [7] and Hall et al. [10]. Because of its simplicity, Phongs' model (see section 3.3) is the commonly used one. It is a reasonable approximation of the specular reflection process [7], based on empirical observations. Substituting (2.2) in (2.1) gives us:

$$I_{out}(\theta_{out}) = E(\theta_{out}) + I_{d,out} + I_{y,out} + I_{r,out}(\theta_{out}) + I_{t,out}(\theta_{out}), \qquad (2.3)$$

where

$$I_{d,out} = k_d\rho_d \int_{\Omega} I_{in}(\theta_{in}) cos(\theta) d\omega, \quad I_{r,out}(\theta_{out}) = k_r \int_{\Omega} \rho_r(\theta_{out},\theta_{in}) I_{in}(\theta_{in}) \cos(\theta) d\omega,$$

$$I_{y,out} = k_y\rho_y \int_{\Omega} I_{in}(\theta_{in}) cos(\theta) d\omega, \quad I_{t,out}(\theta_{out}) = k_t \int_{\Omega} \rho_t(\theta_{out},\theta_{in}) I_{in}(\theta_{in}) \cos(\theta) d\omega.$$

Although we may separate the reflectance into diffuse and specular terms, we can not compute diffuse intensity components $I_{d,out}$ and $I_{y,out}$ independently of the specular intensity components $I_{r,out}(\theta_{out})$ and $I_{t,out}(\theta_{out})$. Indeed, an incoming intensity $I(\theta_{in})$ is just an outgoing intensity $I_{out}(\theta_{out})$ of another patch which in turn can contain both diffuse and specular components. In the preprocess, the effect of incoming specular intensities is accounted for by defining what we called indirect form-factors, which will be illustrated in section 3.3.

Let us assume that all diffuse surfaces in an environment are subdivided into patches, in such a way that each patch exhibits a constant intensity. Then for each diffuse patch pair the geometrical relation (form-factor) representing the fraction of energy leaving a patch and landing on another patch can be drawn up. The form-factor between a differential area dA_i of patch i and another patch j, see figure 1, is defined [5] as:

$$D_{dA_iA_j} = \int_{A_j} \frac{\cos\phi_i \cos\phi_j}{\pi r^2} HID\,(i,j)\, dA_j, \tag{2.4}$$

where

$HID\,(i,j) = 1$ if patch i sees patch j and 0 otherwise.

The form-factor between two diffuse patches i and j is found by taking the area average of (2.4) and is given by:

$$D_{A_iA_j} = \frac{1}{A_i}\int_{A_i}\int_{A_j} \frac{\cos\phi_i \cos\phi_j}{\pi r^2} HID\,(i,j)\, dA_j dA_i \tag{2.5}$$

If the distance between the two patches is large enough, then the integrand of the inner integral in (2.5) remains almost constant [5], hence (2.4) can be used instead of (2.5) to compute the form-factor. Once the form-factors between all diffuse patches have been determined, a set of linear equations can be deduced. In matrix form they look as follows:

$$\mathbf{F}_{R,G,B} \cdot \mathbf{I}_{R,G,B} = \mathbf{E}_{R,G,B}, \tag{2.6}$$

where

$\mathbf{I}_{R,G,B}$ is the vector of intensities,
$\mathbf{E}_{R,G,B}$ is the vector of light source intensities,
$\mathbf{F}_{R,G,B}$ is the form-factor matrix.

Each entry $\mathbf{F}_{R,G,B}$ consists of the product of a form-factor and a diffusion factor ($k_d\rho_d$ or $k_y\rho_y$) of the corresponding patch pair. This formula represents the radiant intensity equation of an environment. The matrix equation can be advantageously solved by using a Gauss-Seidel iteration method.

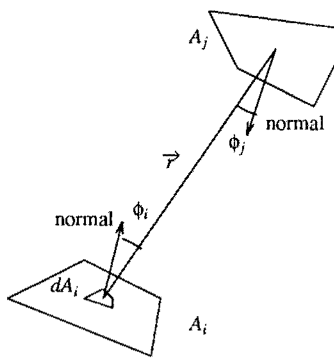


Figure 1. Form-factor geometry.

3. Hemisphere and Form-factors

3.1 Form-factors Between Diffuse Patches: Direct Contributions

As stated in the previous section, a form-factor represents the fraction of energy leaving one surface and landing on another. The *direct* form-factor specifies the direct energy transfer between two diffuse patches and is proportional to the projection of one patch onto a hemisphere placed on the other patch (figure 2a). The direct form-factor from diffuse patch i to diffuse patch j can be defined as:

$$D_{ij} = \frac{1}{\pi}\int_{\xi}\int_{\eta} G'(\eta,\xi)\ \cos\xi d\eta dxi,$$

which is a spherical coordinate equation to (2.4), for r=1.

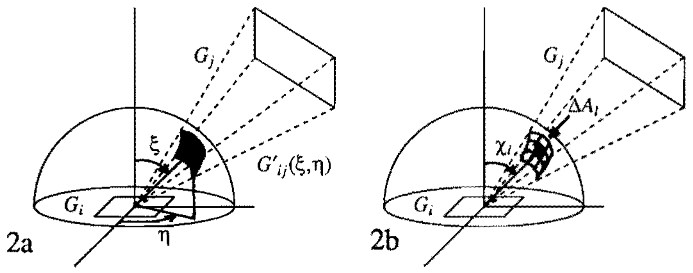


Figure 2. Hemisphere projection (a) and hemisphere discretization (b).

Discretization of the hemisphere into constant differential areas ΔA, figure 2b, yields:

$$D_{ij} = \lim_{\Delta A_l \to 0} \frac{1}{\pi} \sum_l \Delta A_l \cos\chi_l \approx \frac{\Delta A}{\pi} \sum_l \cos\chi_l = K^{-1} \sum_l \cos\chi_l,$$

for ΔA sufficiently small. The normalization factor $K = \Delta A^{-1}\pi$ results from the fact that the fraction of energy emitted within the entire hemisphere equals 1. Hence,

$$K = \frac{\pi}{\Delta A} = \sum_{k=1}^{N} \cos\chi_k,$$

where N is the total number of differential areas on the hemisphere. Each differential area ΔA_l, angle, χ_l thus has its own contribution to the form-factor, called delta form-factor $\delta D_{ij} = K^{-1} \cos\chi_l$.

3.2 Hemisphere Discretization

Although a uniform discretization of the surface of the hemisphere is not easy and does not lead to a uniform distribution of rays defining the hemisphere, a simple and efficient, yet effective method is the following. The rays are such directed that they intersect the hemisphere on circles parallel to the xy-plane with polar angle $j\Delta\theta$, $\Delta\theta$= constant and j= 0,1,...,N, see figure 3.

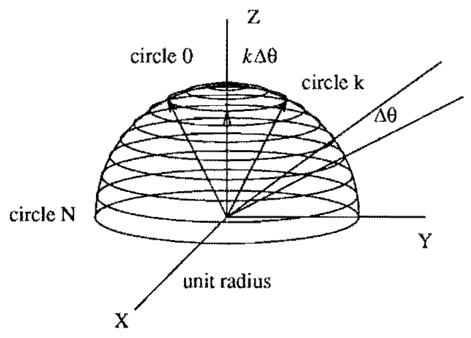


Figure 3. Hemisphere circles

Let the number of rays intersecting circle N, which lies in the XY plane, be equal to M. The distance between two adjacent rays within this circle is then $\Delta x = 2\pi M^{-1}$. We can also make the distance between two adjacent circles equal to Δx, by subdividing the hemisphere into $M/4$ circles, which makes $\Delta\theta = 2\pi M^{-1}$. Two adjacent rays within an arbitrary circle j with angle $j\Delta\theta$ (see figure 3) should then also be spaced by that same distance Δx. The number of rays through circle j must therefore be approximately equal to

$$N_j = \text{round}(M \sin(j\Delta\theta)). \qquad (3.1)$$

The distance Δx_j between two adjacent rays through circle j is then

$$\Delta x_j = \frac{2\pi \sin(j\Delta\theta)}{\mathrm{round}(M(\sin(j\Delta\theta))}.$$

Once the rays that define the quantized hemisphere are determined, their individual contribution is computed and stored in a look-up table. Notice that all rays lying on a single circle have the same delta form-factor. Hence the number of contributions that has to be stored equals the number of circles on the hemisphere. The same hemisphere can also be used to define the frustums that are needed to model directional reflection functions. This will be exposed in the next section.

Remark: By rounding off the number of rays through circle j, $M\sin(j\Delta\theta)$, to an integer that is a multiple of four, only one quarter of the entire hemisphere has to be stored. The error induced by this quantization becomes smaller as the number of rays increases.

3.3 Indirect Contributions to Form-factors

In this section we show how extra form-factors can be computed by taking multiple specular reflections into account, as suggested by Wallace et al. in [20], without computing specular patch intensities. In [20] the form-factor was defined as the quantity specifying the total energy transfer between two diffuse patches. However, this transfer can take place via a strictly direct route (if there are no obstacles) as well as via specular reflections, see figure 4. Consequently, we may write the total received energy of a patch as the sum of the energy received via direct and indirect, specular reflective, paths (see figure 4). These are represented by a direct (D_{ij}) and a indirect (S_{ij}) form-factor contributions respectively, thus:

$$F_{ij} = D_{ij} + S_{ij} = D_{ij} + \sum_{p=1}^{path} S_{ij}^{p}$$

where

F_{ij} is the total form-factor.
D_{ij} stands for the direct energy transfer from diffuse patch i to diffuse patch j,
S_{ij} stands for the energy transfer via (multiple) specular reflection(s),
path is the total number of indirect routes along which energy transfer takes place.

The computation of D_{ij} has been discussed in section 3.1. Here we shall put emphasis on S_{ij}, and we will confine ourselves to a practical example. Suppose that patches 3 and 4 reflect light according to Phong's reflectance function [16] which is given by $(\cos(\alpha)^n)$, where n is the reflection power and α is an angle between the actual reflection direction and the mirror direction. The specular

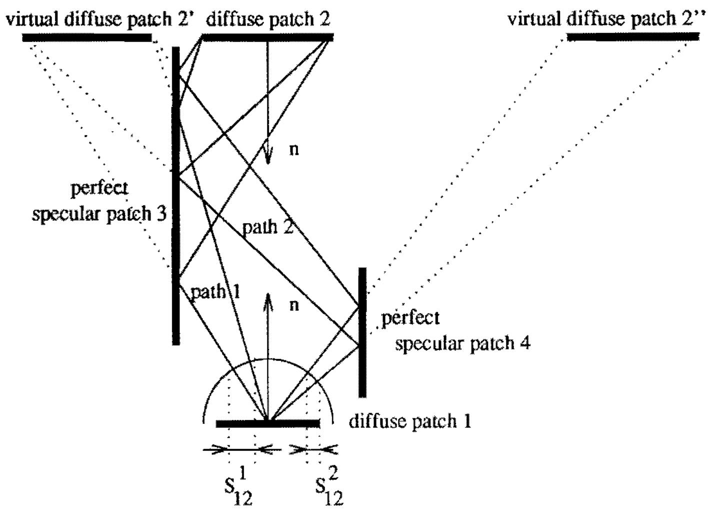


Figure 4. Fractions of the form-factor via perfect specular patches.

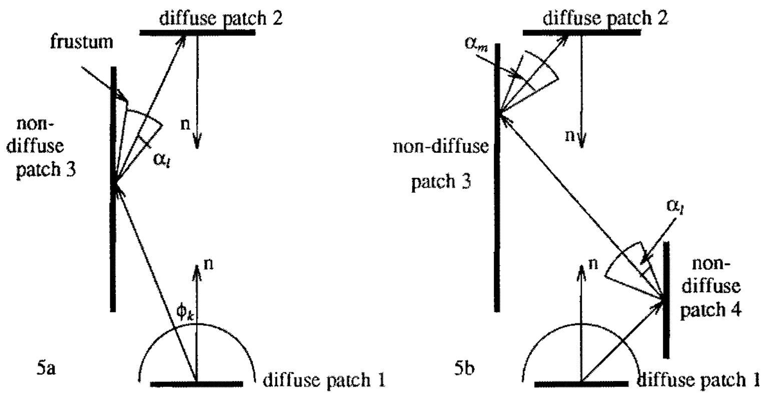


Figure 5. Fractions of the form-factor via non-diffuse patches.

reflectance of a patch may vary from almost diffusely ($n>1$) to perfect mirroring ($n \rightarrow \infty$). To model such a reflectance, a so-called view-frustum is used which in essence is a cone around the z-axis of the hemisphere in figure 3 and defined by a corresponding number of rays from the hemisphere. The frustum size can be adapted to a particular patch reflection low by choosing an appropriate number of rays from the hemisphere. A reflection frustum is directed in the correct position by rotating the rays confined to the frustum. For a given reflection power n, each frustum ray is weighted by a factor, which depends on the reflection angle α.

The indirect form-factors, e.g. between patch 1 and 2 in figure 5a, can be formulated by simply tracing a ray from the hemisphere (patch 1) until it arrives at diffuse patch 2 after a number of specular reflections. For each hemisphere ray intersecting specular patch 3, a frustum of rays will be cast in the reflection direction. Then the contribution of a hemisphere ray to the indirect form-factor S_{12} in path 1, will be multiplied by a factor whose magnitude depends on the position of the ray within the frustum and the number of frustum rays that hits patch 2. This factor is equal to 1 if all frustum rays hit patch 2 and 0 if all rays miss it. Assuming that patch 3 has a reflection power n_3, the additional form-factor between patches 1 and 2 via one reflection (path 1) as shown in figure 5a, is given by:

$$S_{12}^1 = K^{-1} \sum_{k=1}^{y} \cos(\phi_k) \left\{ k_{s3} K'^{-1} \sum_{l=1}^{y_k'} (\cos(\alpha_l))^{n_3} \right\}$$

where

y is the number of hemisphere rays that hit patch 3,
y_k' is the number of frustum rays that hit patch 2,
K'^{-1} is the normalization factor of the frustum.

K'^{-1} results from the observation that the incoming light energy is equal to the scattered light energy in the outgoing direction (if there is no absorption). Similarly, the indirect form-factor via two reflections (path 2), see figure 5b, can be written as:

$$S_{12}^2 = K^{-1} \sum_{k=1}^{z} \cos(\phi_k) \left\{ k_{s4} K'^{-1} \sum_{l=1}^{z_k'} (\cos(\alpha_l))^{n_4} \left\{ k_{s3} K''^{-1} \sum_{m=1}^{z_{kl}''} (\cos(\alpha_m))^{n_3} \right\} \right\}$$

where

z is the number of hemisphere rays that hit patch 4,
z_k' is the number of frustum rays on patch 4 that hit patch 2 via patch 3,
z_{kl}'' is the number of frustum rays on patch 3 that hit patch 2,
K'^{-1} is patch 4 frustum normalization factor.
K''^{-1} is patch 3 frustum normalization factor.

As mentioned before, we can see frustums as part of the hemisphere. Now we shall discuss the computation and the storage of the weight-factors that are assigned to the frustum rays. Consider the hemisphere in figure 3, with N circles on its surface: j=0,1,...,$N-1$. (The circle with number N is not relevant because of the zero contribution.) Then it is possible to construct $N-1$ (j=0,1,...,$N-2$) frustums with different sizes. Observing that all rays lying on a single circle have equal contributions, only one weight-factor for each circle has to be computed. Hence the storage capacity can be reduced considerably.

The circle 0, which has a zero radius, is represented by one ray, i.e. from (3.1), $N_0=1$. For circle j the number of rays N_j is given by (3.1). The weight-factors are computed by discretizing the Phong model in accordance with the hemisphere discretization. The reflection power and angle then become $n = n_k$ (k=0,1,...,$N-2$) and $\alpha = j\Delta\theta$ (j=0,1,...,$N-2$), respectively. The weight-factor for circle j within a frustum with size k is then:

$$W_{jk} = \frac{(\cos(j\Delta\theta))^{n_k}}{\sum_{i=0}^{k}(\cos(i\Delta\theta))^{n_k}\cdot N_i}$$

where $0 \le k \le N-2$ and $0 \le j \le k$.

The denominator provides for the normalization. Using this formula, a weight-factor table can be set up, as follows:

frustum size k / circle j	0	1	2	$N-2$	$N-1$
0	1	W_{01}	$W_{0,2}$	$W_{0,N-2}$	$W_{0,N-1}$
1	0	$W_{1,1}$	$W_{1,2}$	$W_{1,N-2}$	$W_{1,N-1}$
2	0	0	$W_{2,2}$	$W_{2,N-2}$	$W_{2,N-1}$
$N-2$	0	0	0	$W_{N-2,N-2}$	$W_{N-2,N-1}$
$N-1$	0	0	0	0	$W_{N-1,N-1}$

Remark: The column $k=N-1$ represents the contribution of the hemisphere rays to the form-factor, which is a limiting case where the frustum (cone) becomes the hemisphere itself.

3.4 Aproximation of Projections by Ray-Rotating and Ray-Tracing

The projection of area G_j (patch j) onto the hemisphere placed on area G_i (patch i) is computed by shooting one ray from the center of patch i through each differential area ΔA_l of the hemisphere (see fig. 2b). If a ray intersects patch j, then its contribution δD_{ij} is added to D_{ij}. In case one ray has intersections with several patches, the contribution of the nearest patch is taken.

Shooting the entire hemisphere rays from an arbitrary patch is done in the following way. The set of hemisphere rays is created in advance, assuming an initial orientation along direction vector $(0\ 0\ 1)^T$. The direction vector (X Y Z) and delta form-factor assigned to rays are stored for each ray. Then the hemisphere is placed on the patch by rotating all hemisphere rays over the angle between the z-axis and the patch normal, using two Cordic rotation processors [2][13]. The Cordic is capable of rotating a 2 dimensional vector (figure 6). The rotation angles θ and ϕ are computed from $(0\ 0\ 1)^T$ and the patch normal $(nx\ ny\ nz)^T$ by another Cordic, which is not indicated in figure 6. $\phi = \arctan(nx^{-1} \cdot ny)$, $\theta = \arctan(nz^{-1}(nx \cdot \cos\phi + ny \cdot \sin\phi))$, and the rotated vector becomes:

$$(X_{\theta,\phi}, Y_{\phi}, Z_{\theta}) = Rot_{\theta,\phi}(XYZ).$$

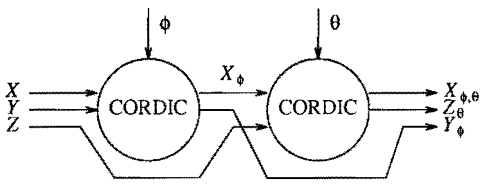


Figure 6. Rotation section

4. Adaptive Patch Sampling

In order to improve image quality, a patch can be subdivided into subpatches (or elements) in case it exhibits a high intensity gradient. In [4] this is done by using an adaptive patch subdivision method. This approach requires adjacency information, which is used to ensure the continuity of the intensity along the element edges in the rendering stage. Rather than subdividing a patch into smaller patches or elements, we compute the center-point of each of these elements, by which the number of patches remains constant. The center-points are obtained by using an "adaptive patch sampling" method. Before an outline of this method can be given, some background information regarding the substructuring technique is given first. For more details we refer to [4,14].

4.1 Substructuring

In the substructuring technique, element-to-patch form-factors instead of element-to-element form-factors are considered. The patch-to-patch form-factor is obtained by taking the area weighted average of the element-to-patch form-factors, that is:

$$F_{ij} = \frac{1}{A_i} \sum_{q=1}^{R} F_{iq,j} \Delta A_{iq} \tag{4.1}$$

where

$F_{iq,j}$ = the form-factor between element q (of patch i) and patch j.
F_{ij} = the form-factor between patch i and patch j.
R = the number of elements on patch i.

The coarse intensities (I_i) of the original patches are computed from the set of equations (2.6). From these intensities, the element intensities are obtained by using element-to-patch form-factors:

$$I_{iq} = F_{iq,j}.I_j \tag{4.2}$$

where

I_{iq} = the intensity of element q of patch i.
I_j = the coarse intensity of patch j.

These element intensities accurately reflect the high intensity gradients on patches. The alternative patch sampling method is described in the next section.

4.2 Patch-Sampling

The idea behind the patch sampling method is based on the fact that actually only the center point of each element and the element-to-patch area ratio are required to compute form-factors. The elements referred to in this section are imaginary elements; their vertices are not relevant during the adaptive patch sampling process. The element center-points (or patch sample points) are defined in a global (x,y,z) coordinate system and their locations on the patch are kept up with local (u,v) coordinates. Both sets of coordinates are determined by using the binary subdivision technique. The ratio of the areas of a element and the patch, specifies the contribution of an element form-factor to the patch form-factor. In fact, this ratio can be interpreted as a weight-factor that can easily be approximated during the subdivision process. Let n be the number of subdivisions needed to obtain a particular imaginary element (q) and let the area of the patch i in the (u,v) coordinate system be normalized to 1, then the weight-factor, W_{iq}, that has to be attached to the corresponding sample point is 2^{-2n}. Hence, formula (4.1) can equivalently be written as:

$$F_{ij} = \sum_{q=1}^{R} W_{iq} F_{iq,j} \tag{4.3}$$

Before starting the patch-sampling procedure, an initial guess must be provided for the number of sample points for each patch on which high intensity gradients are expected. From these points, form-factor computations are carried out assuming (temporarily) that the intensity of an area ΔA_{iq} surrounding sample point q of patch i is constant. For all patches and for all sample points, the form-factors $F_{iq,j}$ are determined. Applying (4.3), (2.6) and (4.2) yields one or more intensity values for each patch. These intensity values are obtained for an initial estimate of the possible gradient that might occur on a patch. To find out whether many samples are needed on a patch, a gradient test algorithm is invoked. This algorithm detects areas with high intensity gradient that have to be sampled finer in a next step. To get the new sample points, a patch is subdivided one step further. After this, the new form-factors for these points are computed and added to the element-to-patch form-factor matrix by removing the old row and inserting the four new rows. Naturally these rows provide more accurate element-to-patch form-factors. Again, applying (4.3), (2.6) and (4.2) will result in more intensity values per patch and hence will lead to a higher image quality. The whole procedure is adaptively repeated until the required criterion is met or the maximum subdivision depth is reached.

In case when there is an extreme high intensity gradient, taking more samples locally is not practical, since the intensity data per patch would increase drastically. A better solution here is to cut the patch along the sharp edge.

4.3 Gradient Test

To perform the gradient test, a 4-sample-point window is placed on the patch and a comparison procedure is activated to find out whether there is a high gradient or not. This procedure locally compares the differences among the intensity values against a prespecified margin. In case of a high intensity gradient, the whole patch is further subdivided to obtain a new set of center-points. If a window lies on a part of the patch with a high intensity gradient, then intensities for these new center-points on that part are computed via form-factor computations. On the other hand, if the window lies on a part of the patch that has no high intensity gradient, then intensities of the new center-points on these parts are obtained via bi-linear interpolation [5] of the intensities of the previous step. Although the subdivision of the whole patch is not necessary in case the gradient is local, it turns out that this procedure not only simplifies the gradient test algorithm, but also provides for a uniform intensity grid which is anyhow desired for the rendering process. In order to minimize storage requirements, one can also store, for each patch intensities hierarchically and perform the interpolations on the fly during the rendering stage.

When the intensity values for the new generated points are known, the gradient test algorithm is applied again. If the required criterion (margin) is not met, the whole procedure will be repeated. At termination, the adaptive patch sampling procedure provides a set of intensity values for each patch on which a high intensity gradient was expected. Finally a grid is placed on each patch such that each mesh of the grid embraces 4 sample points, and grid vertex intensities are derived from the sample point intensities by interpolation. Only vertex intensities are stored; this reduces overhead. Moreover, in the rendering stage interpolation for pixel intensity computation, is now done between the vertices; this is simpler than between sample-points.

5. Rendering

The preprocess provides an intensity grid for each patch. The resolution of the grid varies depending on the intensity gradients on patches.

The postprocess computes the actual pixel intensities for red, green and blue by tracing one ray through each pixel of the displaying screen. If the ray hits a patch, two different (u,v) coordinates are reported to the host. The first (u,v) pair is needed to indicate in which mesh an intersection point is located, and the second pair is needed to determine the actual intersection point within such a mesh. Then, to compute the diffuse intensity component for a screen pixel, we can interpolate between intensities of the corresponding mesh points by using both (u,v) coordinates. If the pixel ray hits a specular patch, then besides the interpolated diffuse intensity component, if any, the specular intensity component is computed by placing a frustum on the patch. If a frustum ray hits a specular patch, a frustum is placed again, and so on. The number of recursive steps can be controlled in two ways. One is by limiting the depth of recursion adaptively according to the amount of the contributed intensity. The other is by reducing the frustum size for successive bounces.

Notice that shooting rays trough the displaying screen is similar to shooting rays from the hemisphere and the computation of specular intensity components corresponds to the computation of indirect form-factors. This means that the same computer architecture can be used for both pre- and postprocess.

6. Hardware

At this stage it is not yet clear which tasks of the entire image rendering algorithm are suitable to be implemented in software or in hardware. As far as the hardware is concerned, we therefore restrict ourselves to the heart of the system which is the computation of form-factors and the solution of a set of equations. All other computations such as bilinear interpolation and adaptive patch-sampling will be assumed to be performed by the host.

Let R be the number of hemisphere rays and P the number of patches within the environment. For a purely diffuse environment a number of $P \cdot R$ rays will be rotated and shot. If space subdivision is not used, all patches must pass each ray, so the number of intersection computations becomes $P^2 \cdot R$, resulting into $P \cdot R$ intersections. The time needed to compute the form-factors is therefore proportional to $P^2 \cdot R$, which makes the intersection computation the most time consuming operation of the algorithm. Specular patches will cause an even greater number of rotation operations and intersection computations.

The rotation operations are performed using a special high-speed Cordic rotation processors [13]. Our goal therefore is to design an architecture built around several intersection computation blocks.

6.1 Intersection Computation

The hemisphere is rotated and placed on each patch. The rays that are shot have the patch center as their origin. Each ray can accordingly be defined as:

$$\vec{r} = \vec{O} + t \cdot \vec{a} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + t \cdot \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}, \text{ with } \|\vec{a}\| = 1.$$

In view of hardware implementation we chose an iterative bounding box based algorithm for the intersection computation, as follows.

```
REPEAT UNTIL STACK IS EMPTY OR ACCURACY CRITERIA ARE MET:
 - POP PATCH FROM STACK
 - SUBDIVIDE PATCH INTO FOUR SUBPATCHES
 - FOR EACH SUBPATCH DO
   BEGIN COMPUTE BOUNDING BOX :
       $MAX_{X,Y,Z}$(CONTROLPOINTS)= $X_{MAX}, Y_{MAX}, Z_{MAX}$
       $MIN_{X,Y,Z}$(CONTROLPOINTS)= $X_{MIN}, Y_{MIN}, Z_{MIN}$
       COMPUTE SIX INTERSECTIONS OF BOUNDING BOX WITH RAY:
         $t_{XMAX}, t_{YMAX}, t_{ZMAX}, t_{XMIN}, t_{YMIN}, t_{ZMIN}$; where $t_{XMAX} = a_x^{-1} \cdot (X_{MAX} - x_0)$
       IF MIN($t_{XMAX}, t_{YMAX}, t_{ZMAX}$)$\geq$ MAX($t_{XMIN}, t_{YMIN}, t_{ZMIN}$) THEN
       hit:=true, PLACE SUBPATCH ON STACK, UPDATE (u,v) COORDINATES
   END.
```

The advantages of this method are:

1. At termination, the algorithm produces the parametric (u,v) coordinates of the intersection point directly.
2. The t-value, that is the distance between the origin of the ray and the intersection point is automatically produced as well as the intersection point itself. The t-value is used for distance comparison, the intersection point for multiple reflection/transmission.
3. The accuracy can be varied by simply adjusting the maximum subdivision level and the minimum box dimension.

The system architecture used for the intersection computation is shown in figure 7, which is similar to the architecture for ray-tracing bicubic patches in Bezier form as proposed by Pulleyblank in [17]. See also section 7.

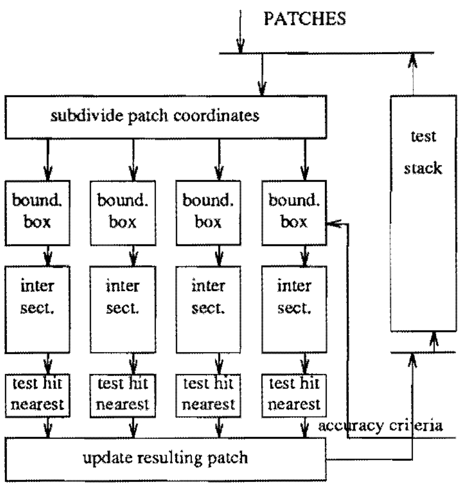


Figure 7. Intersection computation

6.2 Multiple Reflectance

As stated in section 3, due to multiple reflection and/or transmission each primary (hemisphere or pixel plane) ray can possibly generate a tree of frustum-rays to be shot. Such a tree could possibly become too large, which makes shooting all rays almost impossible. This problem is naturally solved using a backtracking procedure. The frustum axes, that is the reflected or transmitted rays that have the largest contribution of all frustum rays, form the tree's skeleton. The skeleton is first composed by only considering the axis of the reflected and/or transmitted frustums. When the contribution of an next axis has become no longer significant after several bounces, the skeleton stops. The last frustum axis is taken from the skeleton and its frustum rays are then shot. It may again be possible for one of these rays to be stored as another frustum axis. If all rays within a frustum are shot, then the frustum that is one level higher in the tree is shot, and so on. This backtracking procedure is carried out until all frustums have been shot for one primary ray. The skeleton must be stored together with all necessary information that is required to generate the frustum rays. The following values have to be stored:

— the frustum axis' origin and direction vector (or direction angles θ and ϕ),

— the frustum size and the number of rays within the frustum that already have been shot

6.3 Functional Block Diagram

A preliminary functional block diagram for the architecture is illustrated in figure 8. The intersection computation block, shown in figure 7, computes the intersection point as described in section 6.1 and passes the required information, such as the number of the intersected patch with its physical description, the intersection point and the contribution (delta form-factor) of the ray. In the k_d processor the direct contributions to the total form-factor are computed. The indirect contributions, via specular reflection and transmission, to the total form-factor are computed in the k_t and k_r processors respectively. If the contribution of the skeleton ray is greater than the threshold value Δ, the transmitted and/or reflected axis ray is computed, pushed on the stack with all necessary information (see sub section 6.2) and immediately shot again. When the contribution of the specular ray(s) is no longer significant, the skeleton is ended. The two circles labeled Cordic provides for the steering of the primary and secondary rays. The whole procedure is supervised by the block labeled control.

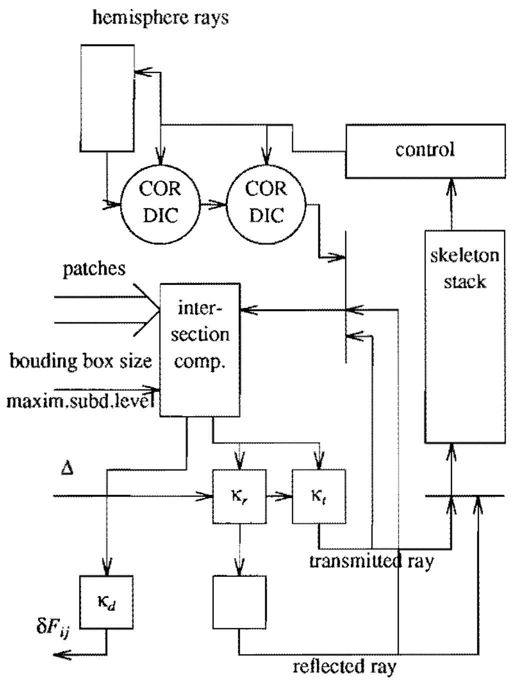


Figure 8. Functional block diagram

6.4 Solving the Radiosity Equations

A less time consuming stage of the algorithm is the solving of the radiosity equations (2.6), using an iterative Gauss-Seidel method. A hardware definition for this stage has already been proposed in [2]. The architecture presented there is built around ASIC processing elements (PEs), which are (pipelined) multiplier/accumulator-processors.

6.5 Hardware Considerations

If the threshold value Δ is kept independent of the frustum size, the error made by the selection of rays to be shot could become very large if the number of rays within one frustum is large. The threshold value Δ should therefore take on a value depending on the number of rays within a frustum. The complexity of the architecture depends on the required accuracy for the intensities of the screen pixel. The screen resolution determines the number of bits needed to express the pixel intensities and the form-factors, which also indirectly determines the number of rays that must be shot from one hemisphere. A thorough investigation in this area is most necessary. Among several further considerations, the most important ones are the following:

— The computation of the reflected and transmitted rays (see figure 8) could be done on board or externally using an extra Cordic-processor.

— The stack for saving the skeleton could become too large to stay on board.

— The subdivision algorithm automatically produces the (u,v) coordinates of the intersection points, which can be used for interpolation when using quadrangular patches. When using different, e.g. triangular patches, another interpolation procedure has to be used.

7. Bezier Patches

A 3D Bezier curve is defined by 16 control points, of which only the 4 corner points actually lie on the surface. A surface representation can be found by subdivision using the control points [17]. After one subdivision step the resulting points define four Bezier sub patches, each determined again by 16 points. The subdivision is repeated until the sub patches may be considered having a constant intensity. In the radiosity method the intensity of each patch is only computed for the center-point of this patch. Without loss of accuracy the leaf patch takes on the intensity value computed for its center-point. In order to compute the form-factors between two arbitrary Bezier patches, we used only 16 center-points and their 16 normals. The hemisphere is placed on the patches centered around each of the 16 points. Let u and v be the parameters along the Bezier patch, we obtain after 3 subdivisions 16 points lying on the surface with (u,v) coordinates $u = 2(n+1/2)/8$, $v = 2(m+1/2)/8$, with $0 \leq m, n \leq 3$. The normal to the surface in such a point is found from the four corner points obtained after 2 subdivisions, by computing the cross product of the two estimated tangent-vectors.

The ray-patch intersection computation is carried out by an iterative bounding box procedure as described in section 6.1. The advantage of this procedure is that the (u,v) coordinates of the intersection point are directly available. The (u,v) coordinates are used to determinate which of the 16 subpatches is hit. The u,v coordinates are also used during the rendering stage, where again a bilinear interpolation procedure is applied. The rendering of shadows across the surface can be accomplished by adaptive patch sampling, similarly as for the polygonal patches (see section 4).

8. Statistics and Simulations

With a software version of the algorithm some simulations have been carried out for various values of the parameters, such as the number of hemisphere rays, number of hemisphere circles and the radiosity of patches. In figure 9a is shown how two radiosity values for RED (b_R), for two arbitrary patches, converge as the number of hemisphere circles R increases. In figure 9b is shown how the total number of hemisphere rays depends on the number of hemisphere circles used. The time needed to compute the form-factors is proportional to the number of hemisphere rays (section 6).

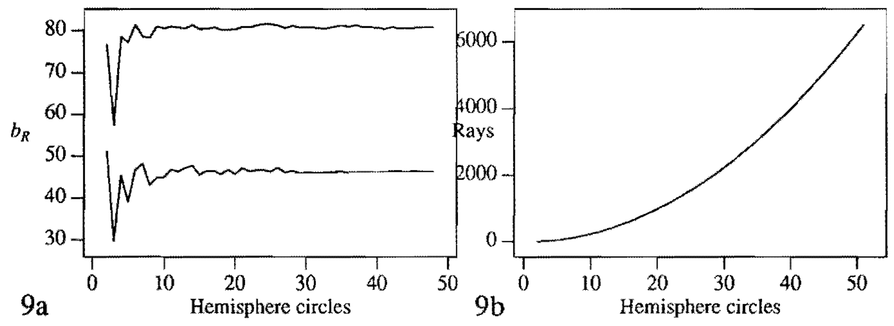
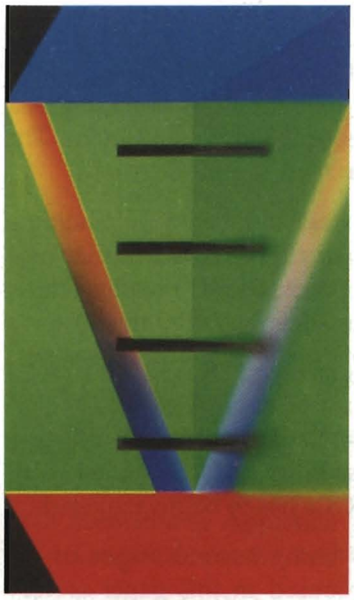
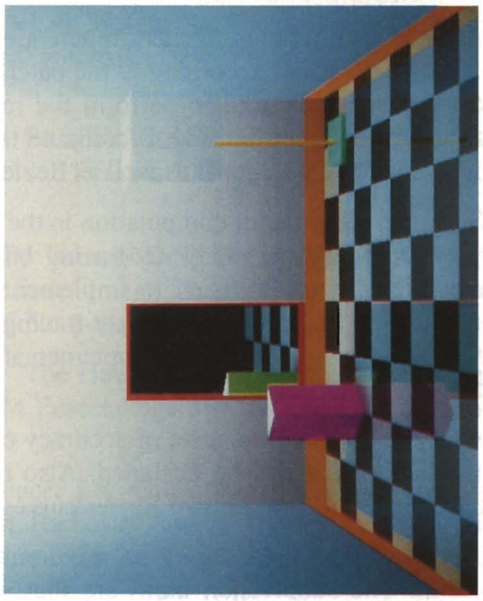
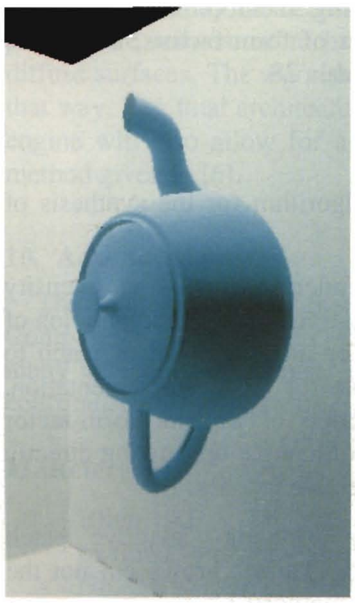


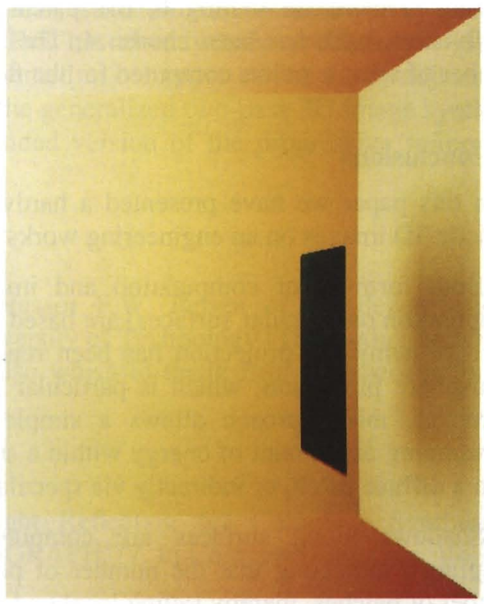
Figure 9. Converging radiosity value (a), and number of hemisphere rays (b).

In picture no.1 a floor is shown that is composed of four different reflecting parts. From left to right : frustum angles: 8,6,4,2 degrees, number of frustum rays: 195,121,61,19. All patches in picture no.2 are purely diffuse and/or perfect mirror reflecting. The two-pass approach proves to be an efficient method to integrate diffuse and specular reflectance behavior. The computation of indirect form-factors accounts for the extra amount of light on the floor directly in front of the mirror. A Bezier teapot consisting of 39 Bezier patches is shown in picture no.3. The number of form-factors determined for this picture is $(39* 16)^2$. An approximation of each Bezier patch by 50 triangular


Picture 1


Picture 2


Picture 3


Picture 4

patches would have required $(39*50)^2$ form-factors to be computed, which implies 13 times as much computation time and 3 times as much memory demand. Picture no.4 enlights the patch sampling method. The maximum subdivision depth has been chosen 4. The number of form-factors, that is the number of sample points computed for the floor equals 28.

9. Conclusions

In this paper we have presented a hardware algorithm for the synthesis of realistic 3D images on an engineering workstation.

Both form-factor computation and image rendering (including intensity computation on specular surfaces) are based on ray-tracing. The computation of patch to hemicube projection has been replaced by an estimation of patch to hemisphere projection, which is particular suitable for ASIC implementation. Moreover, this approach allows a simple selection of a delta form-factor representing an amount of energy within a small solid angle originating directly from a diffuse patch, or indirectly via specular reflective patches.

Shadows along surfaces are computed by applying "adaptive patch sampling", indicating that the number of points on patches grows but not the number of patches, thereby saving local and global memory.

A set of rays defining the hemisphere are stored in advance. For each diffuse patch they are rotated according to the patch orientation and cast independently. Cordic arithmetic is used to perform the rotation operations. A bounding box subdivision procedure is used to compute the ray-patch intersection point. The patches may have polygonal as well as Bezier forms.

Since the form-factor computation in the preprocess and the image rendering in the postprocess are solely ray-tracing based, a single SIMD/wavefront-type architecture is naturally used to implement both processes. This architecture, when built on ray-rotating and ray-tracing pipelined processors allows for a very high throughput VLSI implementation of the generalized two-pass procedure.

As mentioned before, a set of accuracy criteria must be decided upon before a final architecture can be designed. Also the possibility and consequences of including non uniform rational Bezier curves (NURB)s has to be investigated.

The (u,v) coordinates play an important role during several stages of the algorithm. The subdivision hardware that was proposed in this paper is very suitable to generate these coordinates, but it is not suitable for pipelining the operations. This problem can be overcome by modifying the subdivision strategy.

Since the size of the form-factor matrix can not grow larger than the available storage capacity, a restriction to the number of patches within environments may be necessary. This restriction can be removed by using a progressive refinement technique. This has already been presented in [6] for diffuse surfaces. The algorithm presented in our paper can be easily adjusted in that way. The final architecture of the generalized two-pass 3D image synthesis engine will also allow for an extended version of the progressive refinement method given in [6].

10. Acknowledgments

The authors wish to thank Professor F.W. Jansen of the department of Computer Sciences at the Delft University of Technology (The Netherlands), for many valuable discussions. They also whish to thank Erik Platzbecker for his contribution to the software.

11. References

1. Blinn, J.F., "Models of Light Reflection for Computer Synthesized pictures," Proceedings of SIGGRAPH'77, In Computer Graphics, Vol. 11, No. 2, 1977, pp. 192-198.
2. Bu J.and E.F.Deprettere, "A VLSI System Architecture for High-Speed Radiative Transfer 3D Image Synthesis", The Visual Computer, Vol. 5 No. 3 June 1989.
3. Bu J. and E.F.Deprettere, "A Parallel VLSI Algorithm for Fast Sparse Matrix Solution by Gauss-Seidel Iteration". Proc. ISCAS '87, Vol. 3, pp.1052-1055, 1987.
4. Cohen, M.F, Greenberg, D.P, Immel, D.S, Boch, P.J, "An Efficient Radiosity Approach for Realistic Image Synthesis", IEEE Computer Graphics & Applications Vol.6 No.2 March 1986 pp.26-35.
5. Cohen, M.F, Greenberg, D.P, "The Hemi-Cube, A Radiosity Solution for Complex Environments", ACM Proceedings of SIGGGRAPH '85 Vol.19 No.3, July 1985 pp.31-40.
6. Cohen, M.F, Chen, S.E, Wallace, J.R, Greenberg, D.P, "A Progressive Refinement Approach to Fast Radiosity Image Generation", Computer Graphics, Vol. 22, No. 4, August 1988, pp. 75-84.
7. Cook, R.L, Porter, T, Carpenter, L, "Distributed Ray-Tracing", Computer Graphics 18,3,pp.137-146, 1984.
8. Foley, J.D. and Dam, van A, "Fundamentals of Computer Graphics". Addison-Wesley Publishing Co., 1882.

9. Goral, Cindy M., Kenneth E. Torrance, Donald P. Greenberg, Bennet Battaile, "Modelling the Interaction of Light Between Diffuse Surfaces", Proceedings of SIGGRAPH'84, In Computer Graphics, Vol. 18, No. 3, July 1984, pp. 213-222.

10. Hall, R.A. and Greenberg, D.P., "A Testbed for Realistic Image Synthesis", IEEE Computer Graphics and Applications, Vol. 3, No. 10, Nov. 1983, pp. 10-20.

11. Immel, David S., Micheal F. Cohen, Donald P. Greenberg, "A Radiosity Method for Non-diffuse Environments", Proceedings of SIGGRAPH'86, In Computer Graphics, Vol. 20, No. 4, Aug. 1986, pp. 133-142.

12. Kajiya, J.T., "The rendering Equation", Computer Graphics, Proceedings Siggraph 86, Vol.20, No.4, August 1986, pp. 143-150.

13. Lange de A.A.J., Hoeven van der A.J., Deprettere E.F., Bu J., "An Optimal Floating-point Pipeline CMOS CORDIC Processor", Proc. ISCAS '89, Vol. 3, pp. 2043-2048, 1988.

14. McGuire W. Gallagher R.H. "Matrix Structural Analysis", John Wiley & Sons, New York, 1979.

15. Min-Zhi Shao, Qun-Sheng Peng, You-Dong Liang, "A New Radiosity Approach by Procedural Refinements for Realistic Image Synthesis", Computer Graphics, Vol. 22, No. 4, August 1988, pp. 93-101.

16. Phong, Bui Tuong, "Illumination for Computer Generated Pictures," Communications of the ACM, Vol. 18, No. 6, June 1975, pp. 311-317.

17. Pulleyblank, R.W, "The Feasibility of a VLSI Chip for Ray Tracing Bicubic Patches", IEEE CG&A, march 1987 pp.33-44.

18. Siegel R. and Howell J.R, "Thermal Radiation Heat Transfer", Hemisphere Publishing Corp., Washington DC.,1981.

19. Sparrow, E.M. and Cess, R.D. "Radiation Heat Transfer", Hemisphere Publishing Corparation Washington DC., 1978.

20. Wallace, J.R, Cohen, M.F, Greenberg, D.P, "A Two Pass Solution to the Rendering Equation - A Synthesis of Ray-Tracing and Radiosity Methods", ACM Proceedings of SIGGRAPH'87 Vol. 21, No. 4, July 1987, pp. 311-320.