Invited presentation   4th EG Workshop on Rendering, Paris, June 1993, p. 27-46.

Realism in Real Time?

Frederik W. Jansen, Delft University of Technology[*]
Alan Chalmers, University of Bristol[**]

Abstract

With the continuous improvement in ray tracing and radiosity algorithms, image synthesis quality has reached the level of photo realism. However, efforts to achieve real-time performances by implementing ray tracing and radiosity algorithms on parallel processors and dedicated hardware have not been very successful so far. Increasing the number of processors introduces a corresponding growth in inter-processor communication. Caching could be effective to reduce communication, if sufficient coherence would be available in subsequent data requests. Unfortunately, standard ray tracing and radiosity do not provide enough coherence. In this paper we review the different parallel approaches and we try to ascertain those issues that are crucial for further improvement. In particular, we will focus on load and data management strategies that effect the amount of data coherence in ray tracing, and on methods to improve ray and object coherence.

Keywords and phrases: rendering, ray tracing, radiosity, graphics hardware, parallel processing, data coherence

1. Introduction

Over the last decade, computer graphics research has been very successful in achieving two goals: real-time display and increased realism in display. Real-time display has been accomplished by implementing the viewing pipeline of the projective depth-buffer hidden-surface algorithm in dedicated hardware, and increased realism has been accomplished by extending the traditional ray tracing algorithm to include also diffuse interreflection and indirect specular reflection. Unfortunately, the combination of both goals, realism in real time, is still waiting to be realised.

Since the introduction of the geometry engine (Clark 1982), the depth-buffer based display systems have shown a steady increase in performance, both in speed and quality. Starting with a display rate of 30,000 polygons per second in the early eighties, current systems are now able to display more than a million polygons per second, allowing display of reasonable complex scenes in real-time (Torborg 1987; Akeley and Jermoluk 1988; Kirk and Voorhies 1990). Although the depth-buffer algorithm, inherently a projective algorithm, is not able to handle optical effects such as shadows, highlights and mirroring reflections in a natural way, several techniques have been developed to enhance the realism by adding textures, anti-aliasing, motion-blur, depth-of-field, etc. (Heaberli and Akeley 1991). Also diffuse interreflection, and area light sources have been incorporated by adding a radiosity pre-processing that subdivides the scene into a mesh of small surface patches and elements, and calculates the exchange of energy between these patches to account for the diffuse interreflection between surfaces (Cohen and Greenberg 1985;

[*] Faculty of Technical Mathematics and Informatics, Julianalaan 132, 2628 BL Delft, The Netherlands. Email: fwj@duticg.twi.tudelft.nl
[**] Department of Computer Science, Queen's Building, University Walk, Bristol BS8 1TR, United Kingdom, Email: alan@compsci.bristol.ac.uk

Nishita and Nakamae 1985). Display of these elements makes it possible to walk through interiors and still maintain a high-degree of shading accuracy (Baum and Winget 1990; Baum et al. 1991).

The accurate representation of shading and shadow gradients is very much dependent on the accuracy of the element mesh. A resolution too high will be too expensive while a resolution too low will not adequately represent the shading gradients. Adaptive meshing techniques have been developed to provide locally a higher resolution to accommodate shading discontinuities (Cohen et al. 1986). Further improvements have been sought in exact meshing techniques that align the boundaries of elements with shading discontinuities (Campbell and Fussell 1990; Heckbert 1992; Lischinski et al. 1992). In this case, however, it is inevitable that *a priori* knowledge about shading discontinuities is available, obtained for instance from projecting surface contours onto these patches. This is a very expensive and complex kind of (object-space-oriented) pre-processing, particularly if curved surfaces are involved. Nevertheless, the pre-processing will be worthwhile if one wants to display a scene in real time with depth-buffer based display hardware for use in walk-through applications (Baum and Winget 1990). Thus, within the paradigm of depth-buffer-based projective display of polygons a whole set of techniques has been developed to increase realism without sacrificing real-time performance. True realism, however, will still prove to be difficult to achieve with the projective approach because of the lack of specular and mirroring reflections and of the limited accuracy of even the most advanced exact meshing and interpolation technique.

The other display paradigm, ray tracing, has always been appreciated for its high-quality rendering capabilities. The initial 'recursive' ray tracing algorithm (Whitted 1980) did model effectively cast shadows and optical effects such as mirroring reflection and transparancy. With stochastic ray tracing, the repertoire of optical effects was further expanded to soft shadows, motion blur and depth-of-focus, and the image quality was improved by anti-aliasing (Cook 1986; Dippé and Wold 1985; Lee et al. 1985; Mitchell 1987). The addition of Monte Carlo sampling techniques to capture also the indirect light has even further increased the realism and accuracy of the illumination calculation (Kajiya 1986), as did improved reflection models (Cook and Torrance 1984) and texture filtering (Heckbert 1986). A radiosity preprocessing has been introduced to ray tracing too (Wallace et al. 1989; Sillion and Puech 1989) to account for the indirect diffuse reflection. The image quality is here less dependent on the accuracy of the mesh because most important shading and shadow continuities can be re-sampled during rendering, either for specular reflection only (Sillion and Puech 1989), most important direct light (Shirley 1990; Chen et al. 1991; Kok and Jansen 1991) or for all direct and indirect light (Rushmeier 1988; Chen et al. 1991). However, the more sampling is done, the more computation times will tend to explode.

Efficiency improving techniques such as adaptive ray tracing (Painter and Sloan 1989) and spatial subdivision techniques (Glassner 1989) are effective, but processing times for complex scenes are still in the order of minutes. For that reason, ray tracing has always been a popular subject for parallel processing and good results have been achieved (Scherson and Caspary 1989; Green and Paddon 1989), but not in the sense that it has brought ray tracing of complex scenes within reach of interactive display. The alternative of designing special VLSI hardware, the popular route for the depth-buffer approach mentioned above, has not been tried so much for ray tracing. The efforts of Kedem and Ellis (1984, 1989) and Pulleyblank and Kapenga (1986, 1987) and the more recent developments by Shen and Deprettere (1992) are the notable exceptions so far.

The major bottleneck in parallel processing appears to be the data communication between processors. Communication can be reduced by use of caching. However, caching is only effective when enough coherence is available in subsequent data requests (Green and

Paddon 1989). Unfortunately, standard ray tracing as such does not provide much coherence between subsequent ray intersection tasks. In this paper we will therefore review different ray tracing algorithms and we will analyse how the amount of coherence can be increased by adapting the order in which ray intersections are scheduled, and by eliminating as much as possible ray intersections that will tend to destroy potential coherence.

The paper is structured as follows: In section 2, the requirements for realistic rendering are summarised, the state-of-the-art in global illumination reviewed, and an outline of a family of ray tracing algorithms with radiosity preprocessing is given. In section 3, the different hardware and parallel processing approaches are discussed and the role of caching in relieving the communication bottlenecks is emphasised. In section 4, several forms of coherence are discussed and in section 5, techniques to improve data coherence in ray tracing and a hybrid task scheduling strategy are proposed. In section 6, we discuss the open issues.

2. Realism in computer graphics

Realism can only be achieved by combining sophisticated modelling and rendering techniques, such as those for modelling curved surfaces, specifying procedural models, applying texture mapping and filtering, light source models, local reflection models (isotropic/anisotropic, diffuse/specular reflection, refraction, absorption, etc.) and global reflection models (interreflection patterns between surfaces, simulation of soft shadows, mirroring reflections and participating media). Although all these subjects are of equal importance, global reflection (global illumination or inter-reflection) is currently considered to be most crucial, in particular in applications for architecture and interior design.

To give an indication of the complexity of the interreflection problem, some of the paths travelled by the light leaving a light source before it reaches the eye are shown in figure 1. The situation is simplified in the sense that surfaces are assumed to be either purely diffuse or purely specular. Path 1 represents the direct diffuse reflection, path 2 the diffuse-specular reflection, path 3 the diffuse-diffuse reflection and path 4 the specular-diffuse reflection. Other possible paths, e.g. only specular (highlight) or specular-specular reflection are not included in the figure.

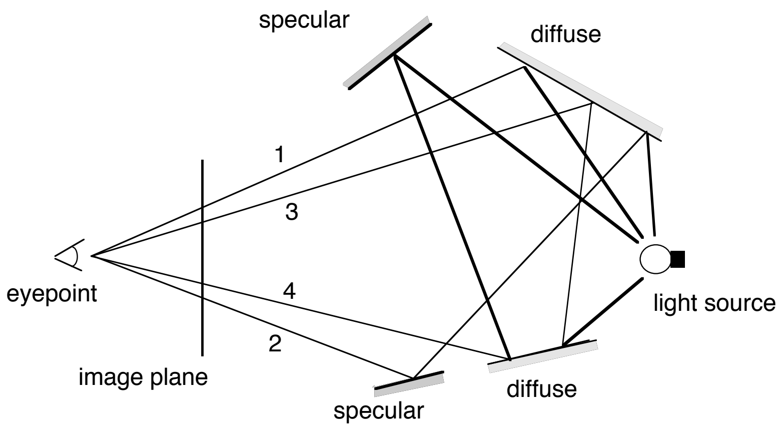

Figure 1. Different paths of light reflection.

Standard projective algorithms (depth-buffer, scan-line, etc.) will only account for light following path 1 and for the direct specular reflection of light, however, without shadow testing. Standard ray tracing (Whitted 1980; Glassner 1989) does sample light following path 1 (including shadow detection) and 2, but it does not account for the indirect reflection of light as a result of the interreflection between surfaces in the scene (path 3), nor for the light that is first reflected by a specular surface before it is diffusely reflected by a visible surface (path 4); see also (Arvo 1986). To capture this light it will be necessary to cast secondary rays into all directions, but now these secondary rays will hit other surfaces for which no intensity is known, and thus the sampling has to be done recursively (Kajiya 1986). Sampling efficiency can be improved by applying importance sampling strategies (Kajiya 1986; Shirley and Wang 1991; Arvo and Kirk 1990) and by exploiting coherence, for instance in the form of illuminance caching (Ward et al. 1988).

To avoid the expensive recursive viewpoint dependent sampling, a viewpoint independent pre-processing - also known as the radiosity pass - can be done to access the global light distribution in a scene and to precompute the amount of light that each surface receives from its environment (Cohen and Greenberg 1985; Nishita and Nakamae 1985). The radiosity pass can be done either by calculating the energy exchange between surfaces in the scene by simultaneous solving a set of linear equations, or by a progressive radiosity method that 'shoots' light from light sources to other surfaces; light which in turn is re-shot to other surfaces, and so on, until a good approximation of the final light distribution is achieved (Cohen et al. 1988). Just as with ray tracing, shooting is preferably done in a stochastic and recursive (but also expensive) way, which is also known as 'particle tracing' (Pattanaik and Mudur 1992).

algorithm	meshing	display	light source sampling	shadow accuracy	time
a	exact/ extensive	ray tracing without spec.	no	dependent on mesh	short
b	extensive	ray tracing	no	dependent on mesh	long
c	moderate	ray tracing	shadow testing primary sources	good	longer
d	low	ray tracing	also secondary sources	better	even longer
e	no/low	ray tracing	recursive sampling	best	longest

Figure 2. Different versions of the two-pass radiosity algorithm.

There are several versions of two-pass algorithms that combine ray tracing-based rendering with a ray tracing-based radiosity pre-processing (Wallace et al. 1987, 1989); see algorithm a, b, c, d, and e in fig. 2. The first (a) displays the pre-computed radiosity values, just as the depth-buffer algorithm. The second version (b) takes the pre-computed radiosity value as the diffuse intensity of the patch and adds the specular reflection component to it by tracing secondary rays (Sillion and Puech 1989). This version still requires an extensive radiosity pre-processing because the shadows from the major (point) light sources are implicitly included in the radiosity shading. The third version (c) only uses the pre-computed radiosity intensity as an improved ambient term and it resamples the light from the most important light sources and patches to calculate more accurate shadows (Shirley 1990; Chen et al. 1991; Kok and Jansen 1991; Kok et al. 1991). This version performs a source selection or source classification during the radiosity pass to determine which patches can be considered as important light sources. The contributions of these selected sources are then not included in the pre-computed radiosity values; during the rendering pass not only specular rays are traced, but also shadow rays are cast to the selected light sources to accurately calculate their contribution to the shading of the patches. The fourth version (d) re-samples all the light by shooting secondary rays to all directions (Rushmeier 1988); now the radiosity shading is not used at all for display, but only to quantify the light that is diffusely reflected by each patch and that is sampled during the rendering by the secondary (shadow) rays.

The last version can be generalised in the aspect that sampling can be continued recursively when the pre-computed radiosity value is not accurate enough (version e). Whether recursion is only done until the first level of recursion, 'one-level path tracing' (Rushmeier 1988), or deeper can be made dependent on the detection of shading discontinuities (highlights, shadow boundaries, etc.) in the neighborhood, or can be a function of the required image quality.

The ultimate algorithm for realism will be a combined radiosity-rendering algorithm that uses a recursive stochastic sampling technique both for shooting (particle tracing) and sampling (ray tracing). With an importance-driven technique (Smits et al. 1992; Pattanaik and Mudur 1993), the radiosity refinement can be focused on the surfaces that are visible. Of course, the amount of ray intersections needed for these algorithms will be too large to be performed in real time by one processor for some time to come, and we will have to defer to parallel processing.

3. Parallel processing

The dedicated hardware and parallel processing approaches that have been so successfully applied to the depth-buffer algorithm, cannot be applied directly to ray tracing, because the ray-object intersection is much more complex than scan conversion and both the objects and rays cannot be processed in a strict linear order suitable for pipe-lining.

In fact, ray tracing can be seen as three quite separate tasks: shading, ray traversal, and ray patch intersection. The shading task initiates primary and secondary rays, performs local light reflection, texture filtering, and anti-aliasing, and is responsible for the final pixel color computation. In the radiosity pre-processing the shading task initiates the hemisphere shooting and updates the patch radiosities. The ray traversal task takes the rays and intersects them with the cells of the spatial subdivision structure, and finds the patches that are candidate for ray intersection. The ray intersection task performs the actual ray-patch intersection and returns the results to the shading task. The shading and ray traversal tasks are very data intensive and not so much computing intensive. The intersection task is both data and processing intensive.

This task breakdown was reason for Gaudet et al. (1988) implementing these tasks separately on micro-coded processors which are connected to each other and to the frame buffer to exchange (intermediate) results. In addition, shading, spatial subdivision, and object data are continuously broadcast over three separate buses. The processors take the information from the buses when they need it and when it comes by. The system can be smoothly scaled up by adding more processors, but communication time will increase proportionally with the size of the object data.

In general, these three different tasks are seldom considered in total. In most cases, the main focus is on the ray traversal and ray intersection task. Dedicated VLSI implementation of the ray intersection task has first been considered by Kedem and Ellis (1984). A proto-type of their ray casting engine for quadratic surfaces has actually been built (Kedem and Ellis, 1989). A design for a VLSI chip for bicubic patch intersection was published by Pulleyblank and Kapenga (1986, 1987). Work on this project has since then been continued and extended to (two-pass) radiosity algorithms (Yilmaz et al. 1989). A design has been made for a 'radiosity engine' in the form of a plug-in board to enhance the performances of standard workstations for high-quality rendering (Shen et al. 1990, 1991). The board contains several intersection processors, each equiped with several ray traversal units. As the basic computational primitive for hardware implementation was chosen the intersection of a frustum (part of a hemisphere) of rays with a set of bicubic patches. Being a more compact surface representation, the communication for bicubic patches is considerably reduced compared to polygons. However, given several very fast pipe-lined computational units to calculate many ray-patch intersections in parallel, the ray-intersection will be extremely fast and the communication between the patch database at the host and the intersection computation units will be the bottleneck. Besides, by exploiting coherence for neighbouring rays (see the following section), reduction in communication is also expected from a hierarchy of caches. See for details and simulation results (Shen and Deprettere 1992).

Many publications have appeared on ray tracing using general purpose parallel processors, in particular using distributed memory systems (e.g. transputer systems), with many showing excellent speedups. However, these good results are usually obtained provided the object data base is replicated on each processor. For complex scenes the data requirements may be very large, far larger than can be accommodated locally at each processor, and now the complete data base must either only be kept by the host and the data sent on request, or it can be distributed over the processors' local memory. This second strategy is preferable due to the communication bottlenecks at the host that are bound to occur with the first strategy.

Assuming that the combined memory of the multiprocessor system is sufficient to contain the whole data base then an initial approach may be to allocate each processor an equal portion of the data base up to the limit of its local memory. Processing load distribution may now be performed in one of two ways. Firstly the tasks can be assigned to the processors that contain the relevant data. In ray tracing, the object data can, for instance, be distributed on the basis of an object space subdivision (Cleary et al. 1983) or an hierarchical box method (Scherson and Caspary 1988). Each processor stores the object data of one (or more) partitions of a spatial subdivision. When a ray intersects one of these portions, the task packet for that ray is sent to this processor and if no intersection occurs or when additional secondary rays are generated then the ray(s) is sent to the processor that stores the appropriate next cell. The object data distribution is thus static and the task distribution follows the 'ray flow' through the system. Although not strictly 'data driven' this approach is often characterised as such. This allocation method will suffer from potential load imbalances should certain areas of the scene attract the majority of the rays. A low-resolution pre-processing can be applied to make a first estimation of the expected load distribution (Salmon and Goldsmith 1988; Priol and Bouatouch 1989) and to correct for this. However, as a result of the different ray directions, the processing load for each partition will vary over time. To keep all the processors busy in such a situation it is necessary to dynamically adjust the size of the partitions and redistribute the object data accordingly. One such system was proposed by Dippé and Swensen (1984) and others (Nemoto and Omachi 1986; I` s, ler et al. 1991) have proposed further improvements. A drawback of these methods is that if the ray flow varies quite some bit (and it mostly does) then re-adjusting the object data does not pay and the efficiency is reduced instead of improved. Kobayashi et al. (1988) try to avoid this by distributing a larger number of space partitions over the processors. In this way, load balancing is better, but the amount of data communication will increase.

The second strategy of work allocation uses a demand-driven approach: tasks are assigned to processors when they are ready to accept new tasks (Green and Paddon 1989; Badouel et al. 1990). The object data required to execute the task is not available locally then it has to be requested from the appropriate remote processor that stores the data. To avoid repeated requests for the same data item, frequently used data items are cached at the local memory of the processor. Part of the memory is thus allocated to store a segment of the data base, the other part is used as a local cache. It is of course preferable to schedule the tasks not completely at random to the processors but instead to take into account the cache contents. Tasks that use the same data should be preferably scheduled to the same processor to allow the cache contents to be re-used. Each processor receives segments of a coherent task and only when other processors run out of their work, segments are assigned to these processors. This can also be implemented by a 'task stealing' strategy. A drawback of the demand-driven approach is that much requested data items will be resident in many caches and much communication is lost on the less used data items. Another drawback is that as the number of processors increases, the amount of communication will grow accordingly.

A hybrid approach is proposed by Scherson and Caspary (1988). Object data is distributed over the processors according to a spatial subdivision or (in their implementation) a bounding box method. Intersection tasks are assigned in a data-driven manner to the processors that contain the relevant data. Ray traversal tasks, however, are assigned in a demand-driven mode: tasks are assigned to processors that are less busy with intersection calculations. This hybrid strategy has the advantage that the intersection computations provide a base load although different for each processor, and that the unbalance in intersection tasks can be compensated with ray traversal tasks, thus avoiding expensive load balancing strategies involving large amounts of data communication. Unfortunately, the ray traversal task is relatively simple and increasing the ray traversal task (by further descending the bounding box hierarchy or by increasing the level of spatial subdivision) increases the amount of communication. Thus, the work load is perfectly balanced and the speed-up is almost linear, but the total performance is low, making this solution less cost effective for a large number of processors.

Summarising, a pure data-driven approach does not seem to be very desirable because of its dependency on the object distribution in space and associated (probably unevenly distributed) processing demand. Far worse, the processing load distribution will probably vary rapidly over time, introducing a severe overhead for dynamic load balancing. Further, because of the large number of rays, the communication overhead for the ray messages will be large. A demand-driven approach, on the other hand, has few problems with load balancing and seems more appropriate for very large data bases. But, here the problem is to keep the data communication as a result of the data requests by the processing tasks as low as possible. As most requests will concern the same (small set) of data items, exploitation of coherence by use of caching is recognised by the proponents of the demand-driven approach as an all important factor for efficiency (Green and Paddon 1989; Badouel et al. 1990; Shen and Deprettere 1992). In the rest of the paper we will discuss this issue in more detail.

For the following discussion, we will partly abstract from the different hardware architectures and consider a system consisting of a high-performance workstation enhanced with a set of intersection processors, either in the form of general-purpose parallel processors or of dedicated VLSI hardware, as the context for our simulation. The workstation itself could be a shared-memory multi-processor system with enough memory to contain the model and shading database. The shading and ray traversal tasks which are assumed to be data intensive and less computing intensive are done on the workstation. The computing intensive ray patch intersection is delegated to the intersection processors. The intersection processors are connected to the host through a controller. The controller is connected to the host with a bus or a fast link and has a relatively large cache. Each intersection processor has his own local memory (fig. 3). The basic idea is very much the same as described in (Green and Paddon 1989) and (Shen and Deprettere 1992).

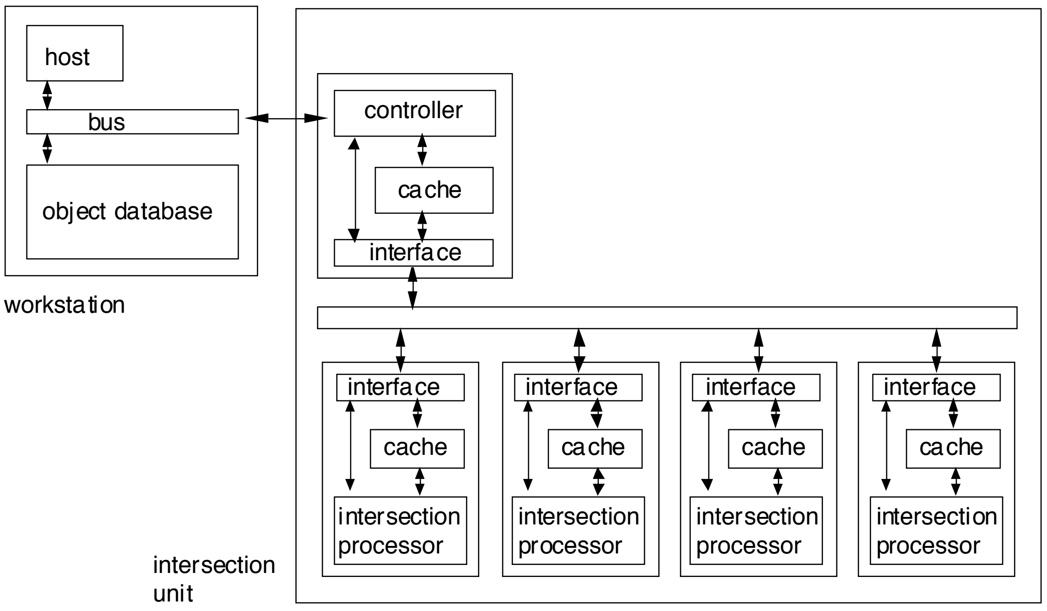

Figure 3. Conceptual architecture (Green and Paddon 1989)

The processor inter-connection could be a bus or a network. With a larger number of processors, the message latency becomes an important factor and a number of techniques have been proposed to avoid a processor standing idle while an object data is fetched from a remote location. Minimum path configurations (AMPs) and message reduction schemes such as 'poaching' and 'snooping' have been shown to be successful in reducing communication overheads and thus the latency for fetching a remote data item (Chalmers 1991; Chalmers and Paddon 1992). Nevertheless, for a large multiprocessor system this delay may still be significant. Multi-threading allows each processor to trace the paths of a number of rays concurrently so that should the computation associated with one ray be delayed awaiting a remote data item, the other computations may still continue (Chalmers et al. 1993). However, it has been shown that the maximum number of threads that can be supported efficiently at each processor is limited and what is worse, this limit is lower for larger number of processors due to increasing message densities.

While the previous methods have been shown to improve the performance of multiprocessor systems, it is nevertheless only in combination with the ideas of caching are we likely to achieve performances on parallel systems approaching the real-time that we desire. Caching is able to exploit any coherence within a scene to significantly reduce the number of remote data item fetches and so once more allow good speed-ups even when the data is distributed across all the processors. Green and Paddon (1989) discuss several caching strategies. Their starting point is a demand-driven approach on basis of image space subdivision. Tasks, assigned to processors, make requests for object data, which are satisfied either by a local resident set, a local cache, the cache of the controller, the resident sets and caches of neighboring processors, or by the host. A low resolution ray tracing is proposed as a profiling method to select the resident set and the initial cache filling. A similar strategy is proposed in (Shen and Deprettere 1992).

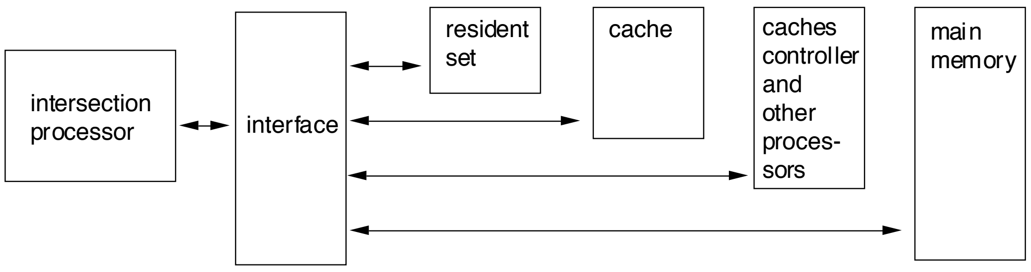

Figure 4. Caching strategy (Green 1991)

The effectiveness of caching is significantly dependent on the coherence that the algorithm can offer, and thus the overall effectiveness of parallel processing is is also significantly dependent on the way the ray tracing and radiosity algorithms proceed and the order in which intersection calculations are scheduled.

4. Coherence

Green and Paddon (1989) discuss several forms of coherence. Beside image and frame coherence they discern the following types:

- ray coherence: neighbouring rays will likely intersect the same surfaces; this allows reuse of patch data used for previous rays (Speer et al. 1985; Hanrahan 1986, Arvo and Kirk 1987).
- object coherence: local neighborhoods of space tend to be occupied by only a small set of the total number of objects; this allows for a fast localization of the candidate set for intersection with a spatial subdivision technique (Glassner 1989)
- data coherence: requests to the object data base will tend to be restricted to a small subset of the whole data base and tend to show a large amount of spatial locality (Green and Paddon 1989).

Within a certain time span, data coherence can be interpreted as a more general form of ray and object coherence: given a sequence of data requests, many requests will be for the same data items. This form of coherence is very much dependent on the order the algorithm proceeds and traverses through object space.

We will now discuss the different forms of coherence in more detail.

4.1 Ray coherence

Clustering neighboring rays into frustums has been proposed earlier to reduce ray tracing costs and to perform efficient anti-aliasing (Speer et al. 1985; Hanrahan 1986). Ray frustums are a very versatile computational primitive within ray tracing and radiosity. The primary rays in ray tracing form a frustum with the eye point as origin. Ray frustums are further used to sample area light sources, and to simulate depth-of-focus and motion blur. In radiosity algorithms, hemispheres of rays are used for shooting.

Intersection of a frustum of rays with a set of patches as a computational primitive is particularly attractive for a pipe-lined hardware intersection unit that can test a bundle of rays simultaneously against a set of patches (Shen et al. 1990). To further increase coherence and to distribute the processing over multiple intersection units, Shen et al. subdivide each frustum into sectors (see fig. 5). All rays in one sector will be loaded on one intersection computation unit. The size (angle) of the sectors is made dependent on the expected patch density and patch distribution to obtain a good load balancing over the different intersection units and to keep the number of calls to the database minimal.

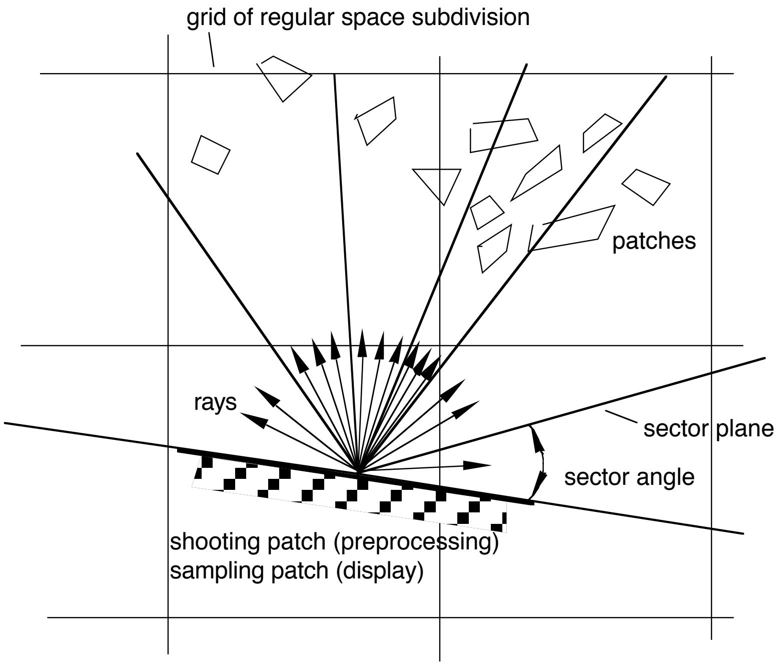
grid of regular space subdivision
patches
rays
sector plane
sector angle
shooting patch (preprocessing)
sampling patch (display)

Figure 5. Ray frustum and sectors.

The ray frustum intersection is indeed a versatile computational primitive for hardware implementation. However, it has also some drawbacks. The ray frustum method is mainly intended for *undirected* shooting and sampling (see fig. 6), which is to avoid the overhead at the host involving searching the whole data base and clustering the rays into sectors. This means that rays are cast without aiming at a specific patch or a specific point (e.g. a vertex). This fits well within a Monte Carlo type of sampling, but not very well within a progressive radiosity method, as it is implemented usually (with directed shooting). See for a discussion on the advantages and disadvantages of directed and undirected sampling (Wallace et al. 1989), and (Shirley and Wang 1992) where these are called implicit and explicit sampling. The undirected sampling poses some additional constraints on the resolution of the mesh with respect to the resolution of the rays.

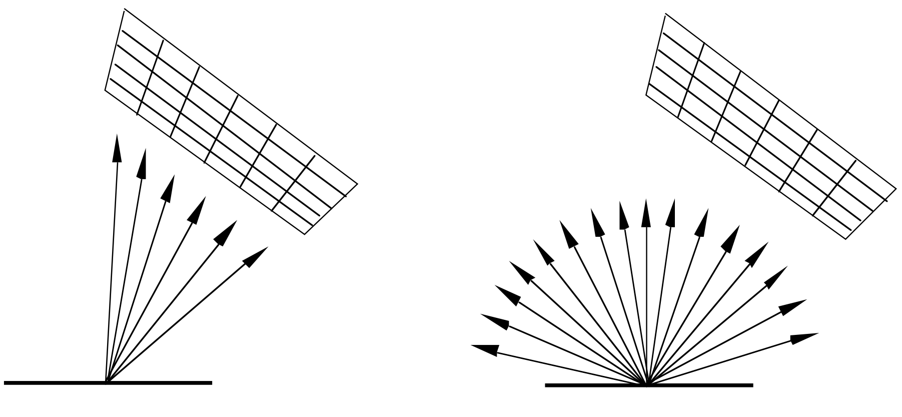

Figure 6. Directed (left) versus undirected shooting (right).

If the number of rays is too low and the mesh resolution too high then some elements of the mesh will not receive a contribution or the contribution will not be spread evenly. This in fact is bound to happen because as the distance over which the rays travel increases, the rays will get further separated and thus the mesh resolution can never be optimally adapted to the ray density. See for instance the disastrous effect of undirected shooting from a light source to a patch that is regularly subdivided into 256 elements (fig. 7). Directed shooting takes 293 rays (fig. 7h right-below). A comparable quality can only be obtained with more than 100,000 rays shot in an undirected way (see fig. 7a-g).

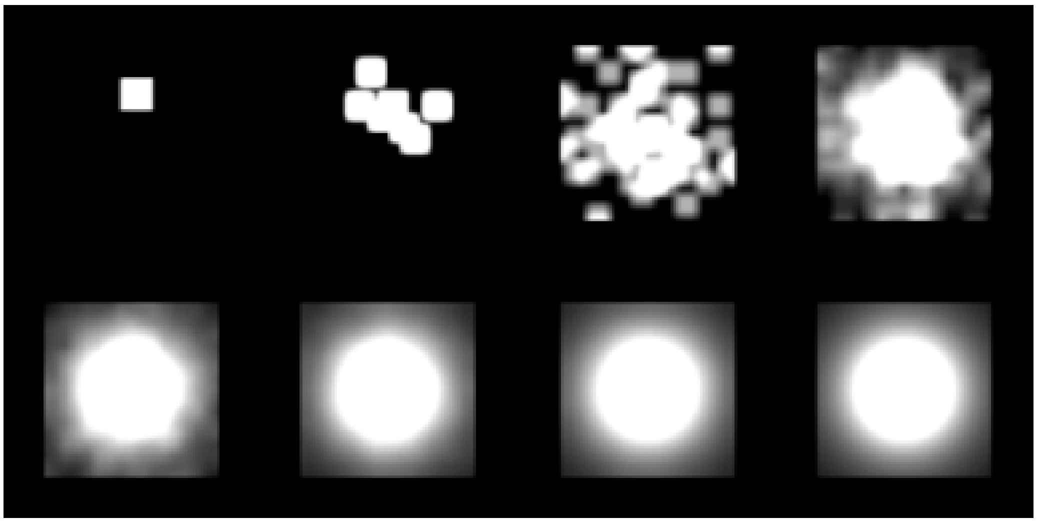

Figure 7. Results of radiosity pass for a patch in front of a light source. Undirected shooting a: 1, b: 10, c: 100, d: 1000, e: 10000, f: 100,000 rays, g: 1,000,000 rays, and directed shooting: h: 293 rays.

To accommodate the resolution of the receiving mesh to the resolution of the rays, an hierarchical mesh data structure should be used that assigns intensity values to levels corresponding to the density of the receiving rays (Asensio 1992; Languénou et al. 1992). At the end of the radiosity pass, the different levels could then be merged to obtain the radiosity values of the patch vertices. Further, the number of rays could be made dependent on the (expected) density of the patches in a sector, the sizes of the patches, the distance of the patches from the ray origin, the intensity of the patches (during display), the chance of shading discontinuities (both during radiosity preprocessing and display), the reflectance properties of the patches, etc. So source selection and source classification criteria could be applied here (Kok and Jansen 1991; Chen et al. 1991). The ray density would then reflect the 'importance' of the shooting/sampling direction of that sector.

Another problem with frustum tracing is that shooting preferably is not done in a hemisphere type of way, but by shooting individual rays from stochastically distributed positions on the patch preferably reflecting the energy distribution over the patch. Ray directions should be stochastically distributed as well to avoid aliasing. Finally, for specular and mirroring reflection, a frustum of rays will spawn a large number of secondary rays, however, with different starting points and with possible different directions. This will all have a dramatic effect on the ray coherence. Therefore, a compromise will have to be sought here between accuracy and coherence.

4.2 Object coherence

The fact that a certain neighborhood of space will tend to contain only a small subset of the total objects in a scene, allows us to localize the candidate set for intersection with a given ray by applying a spatial subdivision technique. Regular grids as well as adaptive space subdivision schemes have been proposed (Glassner 1989). A uniform grid allows for a fast ray traversal, but does not adapt well to an uneven distribution of objects. A two-level grid where complex objects have their own internal grid, seems to combine the advantages of fast traversal and adaptive space subdivision. Unfortunately, for a frustum of rays the spatial subdivision is not nearly so effective as for single rays. Only near to the frustum origin, object localization is achieved. As the rays propagate through space, the distance between the rays increases and ray coherence diminishes. This is amplified by the fact that some of the rays will be intercepted by patches while others continue. Object coherence is also small for objects consisting of a large number of small patches (for instance a keyboard with individually modeled keys). Each ray will hit another patch. Again, as with the shooting resolution, the level of object intersection has to be adapted to the density of the rays. A large number of small patches should be grouped together if this set of patches is only intersected by a small number of rays (Kok 1993). Further, ray tracing should be stopped when the coherence drops below a threshold value. Rays could be assigned values based on the average intensity of a group or subspace. Like in the virtual walls techniques (Xu et al. 1989), planes of the spatial subdivision structure could be given intensity values representing the scene behind them. Rays that would loose coherence could be terminated by assigning them values from this object or space hierarchy.

4.3 Data coherence

For the more general form of coherence in subsequent data requests, Green and Paddon (1989) keyed the term data coherence. This type of coherence is caused by repetition in the algorithm, for instance by shooting multiple hemispheres from the same patch, or by repeated shadow testing for the same light sources. In general this type of coherence is very much dependent on the order the algorithm proceeds, in particular, on the order rays trees are traversed through space. Each primary ray will spawn, beside shadow rays, a number of secondary rays to account for specular reflection and transparency. Each of these secondary rays might generate in turn a number of tertiary rays, and so forth. The amount of coherence in the primary rays will be large, but will drop quickly with each new generation of rays (see fig. 8).

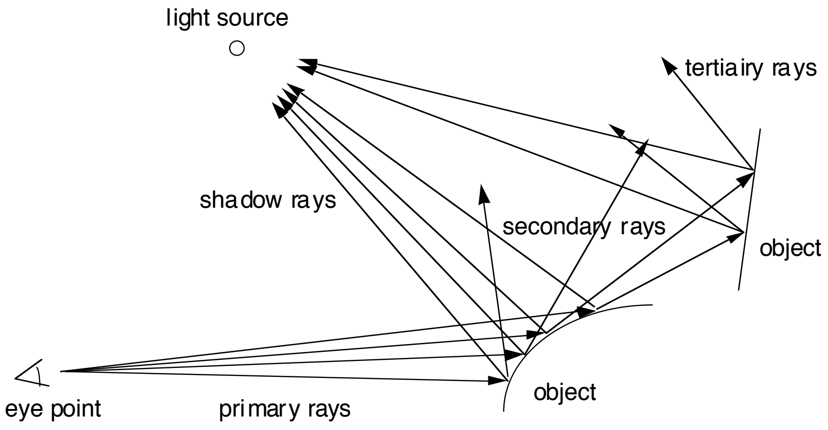

Figure 8. Ray tree

However, if the density of the primary rays increases - say from 4 per pixel to 64 per pixel- then the coherence for the primary rays will improve only slightly (because it is already high), but the coherence for the secondary and following rays will grow considerably. This notion is the basic idea behind the adaptive rendering algorithm proposed in the following section.

Note also that a 'breadth-first' shooting of the ray tree may be preferably over 'depth-first' shooting of the ray tree (Hanrahan 1986). Breadth-first shooting will require that intermediate results will have to be temporarily stored. The same would be true for methods that would try to benefit from coherence between neighbouring ray trees, for instance to shoot simultaneously to the same light source. In general, a breadth-first approach, i.e. first shooting all the primary rays, then all the secondary rays, etc. would be beneficial to the coherence. Of course, in practice this strategy cannot be fully exploited because of the limitations to the amount of processing and storage that will be needed for the intermediate results. Note however, that in the progressive radiosity this actually is done: ray recursion is only applied for specular reflection. Diffuse reflection is postponed until a patch becomes a shooting patch. This can be accomplished because the diffuse reflection is more or less direction independent. Recursive shooting (particle tracing) can be characterised as a depth-first approach.

Finally, a more general form of spatial coherence could also be exploited. For instance, in progressive radiosity, speed of convergence could be sacrificed to optimise coherence resulting from shooting from neighbouring patches, i.e. patches near to each other are processed directly after each other, ignoring for a while patches with a higher priority at a larger distance.

5. Coherence improving methods

At this point we will bring together the different components of our analysis. Our main concern is a ray tracing algorithm with a radiosity pre-processing and with additional shadow sampling for the main light sources to improve shadow accuracy (see section 2). The algorithm will be implemented on a high-performance workstation enhanced with several intersection processors. Data communication between host and intersection processors is reduced by use of caching (section 3). Nevertheless, the communication between the host and the intersection processors will be considered to be the bottleneck in the system (and not the shading calculations or ray traversal calculations by the host), and thus the performance of the system will be determined mainly by the appearance of coherence in subsequent patch requests (section 4). What we want to achieve is that the amount of communication will be kept in proportion to the amount of intersection computations, that is, the coherence should be kept at a constant, preferably high, level.

There are several ways to achieve this: by choosing a suitable breadth- or depth-first ray tracing method, by improving object coherence with an hierarchical object data base, and by a task and management strategy that optimally matches processing and data demands.

5.1 Recursive depth and sampling control

We could imagine different modes for the rendering (given an already calculated radiosity pre-processing) in the same style as the adaptive refinement method of Chen et al. (1991):

- direct display of the radiosity mesh; only primary rays are cast and shading is directly interpolated from the pre-calculated radiosity values; only the direct visible patches will be communicated to the intersection unit

- ray tracing with shadow testing; shadow testing is added to improve shadow accuracy for the most important light sources; the intersection processors will now request also patches in between the visible patches and the light sources

- ray tracing with secondary rays for simulating mirroring reflections and transparency; each of the secondary will in its turn also spawn shadow rays

- ray tracing with diffuse and specular inter-reflection; secondary rays are cast into all directions to sample the indirect light; at this stage we use the pre-computed radiosity shading only to answer these ray queries; in some cases, if required, sampling may be continued for another recursive level

- in addition, a continuous shooting process could refine the radiosity solution, in particular for the visible part of the scene. Also here a trade-off can be made between a simple progressive radiosity pass or a more recursive, stochastic method.

It is clear that the first mode (only primary rays) provides the highest degree of coherence, and that coherence will be minimal for the most elaborate form of sampling for the diffuse and specular reflection. However, as the initial sampling rate of the primary rays increases, the coherence will increase as well, in particular for the secondary and tertiary rays, and the more ray recursion can be allowed, given a pre-specified amount of coherence (to balance the computation and communication). The basic idea, therefore, is to link the level of ray recursion to the amount of available coherence. The coherence in its turn is dependent on the initial chosen sampling rate of the primary rays and on local patch parameters such as the curvature of the surface, the number of relevant light sources, etc. Decisions to continue ray tracing or to start another level of ray recursion should, therefore, preferably be made based on local criteria.

The algorithm could be implemented as follows. As the primary rays hit a patch, the intersections are collected on a patch by patch basis. Secondary and shadow rays are calculated for these points and the amount of coherence is estimated on basis of the specular coefficient, the number of rays, the position and direction of the rays, etc. Only if the coherence is high enough then (frustums of) rays are sent to the intersection processors. Otherwise, ray tracing will be deferred until enough rays have been collected, or alternatively, a pre-processed radiosity/ambient value will be assigned as an estimate for the correct value.

Further, each ray will only be continued as long as there is enough coherence; if the coherence drops below a certain level then the ray will be terminated and assigned a value from the virtual database (e.g. the cell planes of the spatial subdivision). Also, a hierarchical element mesh structure or a hierarchical object grouping will be applied to adapt the sampling resolution of the rays to the size of the elements, patches and objects.

5.2 Task and data management

So far, the proposed coherence techniques do not specify how the intersection tasks are assigned to the processors. It will be clear that the primary rays within one frustum (or section of a frustum) will preferably be assigned to the same processor. This may also hold for some of the secondary and shadow rays. At a certain point, however, ray and object coherence drops below a certain level and only a more general form of spatial coherence which is more difficult and expensive to grasp (e.g. the overlap in visible patches seen from different, possible remote, patches), is left.

When coherence becomes low, it means that data communication will grow. This may be to the amount that 'data flow' communication will exceed the amount of 'ray flow' communication that would occur with a pure data-driven approach. Therefore, we could imagine a hybrid approach in line with (Scherson and Caspary 1988) where each processor would store a partition of the spatial subdivision and process the (random-oriented) rays that traverse that partition. The more coherent ray frustums could then, in a demand-driven mode, be assigned to processors that are less busy with their own partition. The result will be that many-referenced items may be replicated over many processors, however, without increasing the data communication for the less-used data items. In case more coherence would be needed for the demand-driven tasks then a specialisation can also be applied: some processors could work purely or mainly in a demand-driven mode and other processors in a data-driven mode.

6. Conclusions and discussion

Data communication will be the main bottleneck in general purpose parallel processing systems with distributed memories for some time to come. In custom-designed systems the communication bandwidth can be optimised, but data transfer rates will still be a limiting factor. Data communication requirements can be relieved with a suitable caching strategy. The effectiveness of caching will depend strongly on the amount of data coherence the algorithm can offer. We have discussed several sources of coherence and proposed some methods to increase object and spatial coherence. We have analysed possible data and task management strategies and proposed a hybrid algorithm that uses both a demand- and data-driven task assignment strategy. Demand-driven will be applied for high-coherent ray intersection tasks and data-driven for low-coherent tasks.

There are several issues to be explored yet. One is the trade-off between regular and stochastic ray tracing. Regular sampling/shooting (standard ray tracing and progressive radiosity) will proceed in incremental order and while ray coherence is optimal, the load balancing may be strongly effected by the direction of the rays. A stochastic sampling approach, on the other hand, casts rays from random positions into random directions and will exhibit less data coherence, but the overall pattern of computation will be more stable. It may then be worthwhile to optimise not only the load balancing, but also the communication flow between processors, for instance, by minimising the average distance of the ray flow. It is not to say in advance whether an optimal load balancing and consistent ray flow will compensate for the loss of coherence. This will have to be verified by experiments.

Ray tracing in combination with particle tracing will provide the biggest challenge. The algorithm can be described as a stochastic depth-first shooting and sampling. It can effectively model all kinds of anisotropic reflection and scattering in participating media. However, coherence will be minimal. This will probably mean that for parallel processing the load balance is perfect, but that ray flow will be large and object data flow large as well, unless local memory will be sufficient to encompass all object data that is relevant for a (semi-permanent) task.

A possible architecture is depicted in fig. 9. There are three clusters of processors, one works in a demand-driven mode, one in a data-driven mode, and one processes the ray tracing results and does the final image synthesis. The demand-driven cluster processes the first generations of rays for both the particle tracing and the rendering. Rendering rays have priority over particle rays, however, when the view point does not change then particle rays may be given a higher priority. As soon as coherence has dropped below a pre-specified level, then the rays are passed to the data-driven cluster. The proposed architecture could be converted into a minimum-path configuration. For instance with a 6x6 data-driven cluster, the maximum distance from any of the demand-driven/image processors to any of the data-driven processors could be minimized to four. Which configuration is optimal will depend very much on what the critical communication paths will be.

In the discussion so far, we have neglected the shading task. Texture mapping and physically-based reflection models may involve quite some data. As long as the shading task can be done by the host, the data can reside at the host. If the shading task has to be distributed, then given the amount of data, it can only be in data-driven mode. The same may apply for the radiosity mesh information.

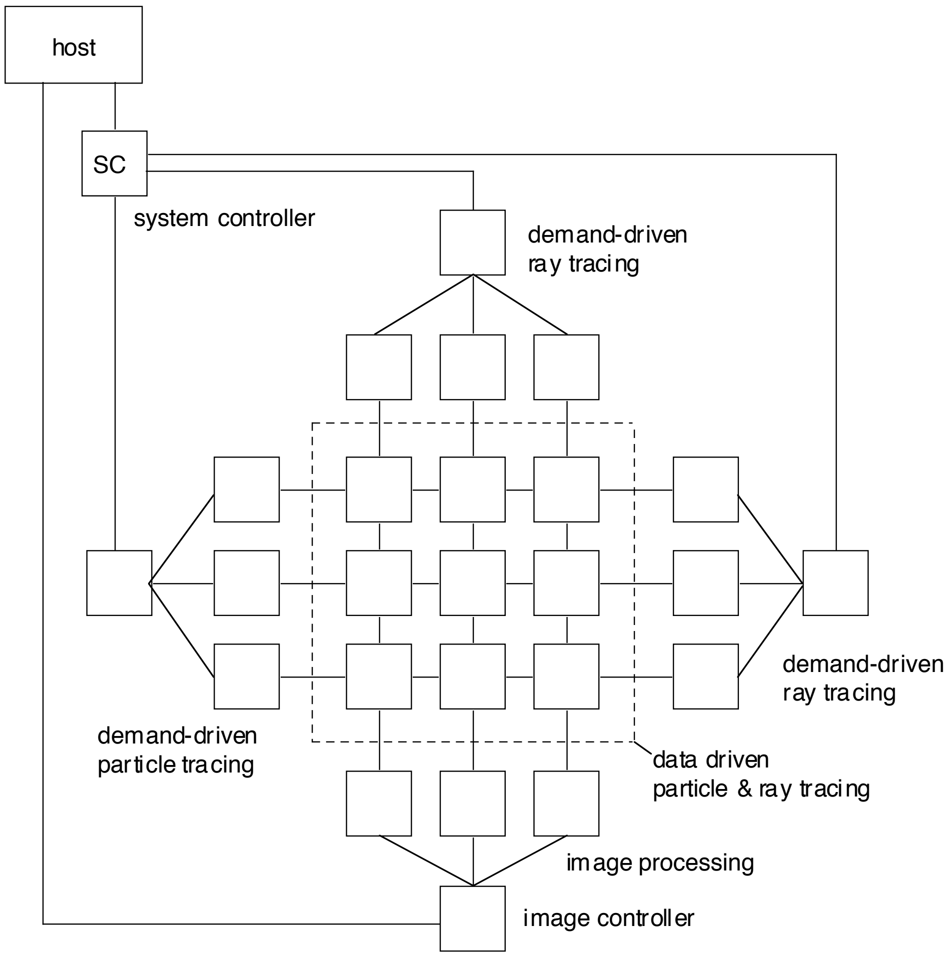

Figure 9. Architecture for combined particle-ray tracing.

Our intention was to review the current state-of-the-art and outline possible directions for further research. A lot of work still has to be done before we will see photo-realistic images of complex scenes being generated in real time.

Acknowledgements

The ideas reported in this paper have grown out of discussions with Arjan Kok and the members of the Radiosity Engine Project, Ed Deprettere, Gerben Hekstra, Li-Sheng Shen and Theo Verelst from the Network Theory Section of the Faculty of Electrical Engineering of Delft University of Technology.

References

Akeley, K., Jermoluk, T. (1988), High-performance Polygon Rendering, Computer Graphics 22(4): 239-246, Siggraph'88.

Arvo, J. (1986), Backward Ray Tracing, Developments in Ray Tracing, Siggraph'86 course notes.

Arvo, J., Kirk, D. (1987), Fast Ray Tracing by Ray Classification, Computer Graphics 21(4):55-64.

Arvo, J., Kirk, D. (1990), Particle Transport and Image Synthesis, Computer Graphics 24(4):63-66, Siggraph'90.

Asensio, F., (1992), A Hierarchical Ray Casting Algorithm for Radiosity Shadows, Proceedings of the 3rd Eurographics Workshop on Rendering, Bristol.

Badouel, D., Bouatouch, K. Priol, T. (1990), Strategies for Distributing Data and Control for Ray-Tracing on Distributed Memory Parallel Computers, Siggraph Course Notes 28, 1990. To appear in Computer Graphics and Applications.

Baum, D.R., Winget, J.M. (1990), Real Time Radiosity Through Parallel Processing and Hardware Acceleration, Computer Graphics 24(2): 67-75.

Baum, D.R., Mann, S., Smith, K.P., Winget, J.M. (1991), Making Radiosity Usable: Automatic Preprocessing and Meshing Techniques for the Generation of Accurate Radiosity Solutions, Computer Graphics 25(4): 51-60.

Campbell, A.T., Fussell, D.S. (1990), Adaptive Mesh Generation for Global Diffuse Illumination, Computer Graphics 24(4): 155-164, Siggraph'90.

Chalmers, A. G. (1991), A Minimum Path System for Parallel Processing. PhD thesis, University of Bristol.

Chalmers, A.G., Paddon, D.J. (1989), Communication Efficient MIMD Configurations. Proc. of the 4th SIAM Conference on Parallel processing for Scientific Computing, Chicago.

Chalmers, A.G., Paddon, D.J. (1991). Parallel Processing in the Progressive Refinement Radiosity Method. Proceedings of the 2nd Eurographics Workshop on Rendering, Barcelona.

Chalmers, A.G., Stuttard, D., Paddon, D.J. (1993), Data Management for Parallel Ray Tracing of Complex Images. International Conference on Computer Graphics, Bombay.

Chen, S.E., Rushmeier, H., Miller, G., Turner, D. (1991), A Progressive Multi-Pass Method for Global Illumination, Computer Graphics 25(4): 165-174, Siggraph'91.

Clark, J. (1982), The Geometric Engine: a VLSI Geometry System for Graphics, Computer Graphics 16(3): 127-133, Siggraph'82.

Cleary, J.G., Wyvill, B., Birtwistle, G., Vatti, R. (1983), Multiprocessor Ray Tracing, Tech. Report 83/128/17, Dept. of Computer Science, Univ. of Calgary; also in Computer Graphics Forum 5(1): 3-12.

Cohen, M.F., Greenberg, D.P. (1985), The Hemi-cube: A Radiosity Solution for Complex Environments, Computer Graphics 19(3): pp 31-40, Siggraph'85.

Cohen, M.F., Greenberg, D.P., Immel, D.S., Brock, P.J. (1986), An Efficient Radiosity Approach for Realistic Image Synthesis, IEEE Computer Graphics and Applications 6(3): 26-35.

Cohen, M.F., Chen, S.E., Wallace, J.R., Greenberg, D.P. (1988), A Progressive Refinement Approach to Fast Radiosity Image Generation, Computer Graphics 22(4): 75-84, Siggraph'88.

Cook, R.L. (1986), Stochastic Sampling in Computer Graphics, ACM Transactions on Graphics 5(1): 51-72.

Cook, R.L, Torrance, K.E. (1982), A Reflectance Model for Computer Graphics, ACM Transactions on Graphics 1(1): 7-24.

Dippé, M., Swensen, J. (1984), An Adaptive Subdivision Algorithm and Parallel Architecture for Realistic Image Synthesis. ACM Computer Graphics 18(3):149-158.

Dippé, M., Wold, E.H. (1985), Antialiasing through Stochastic Sampling, Computer Graphics 19(3): 69-78, Siggraph'85.

Gaudet, S., Hobson, R., Chilka, P., Calvert, T. (1988), Multiprocessor Experiments for High-Speed Ray Tracing, Transactions on Graphics 7(3):151-179.

Glassner, A.S. (1989), Introduction to Ray Tracing, Academic Press.

Green, S., Paddon, D. (1989), Exploiting Coherence for Multiprocessor Ray Tracing. Computer graphics and Applications, 4(10):15-22.

Green, S. (1991), Parallel Processing for Computer Graphics. Research Monographs in Parallel and Distributed Computing, Pitman Publishing, London.

Hanrahan, P. (1986), Using Caching and Breadth First Search to Speed Up Ray Tracing, Proc. Graphics Interface'86, Canadian Information Processing Society, Toronto, pp. 56-61.

Heaberli, P., Akeley, K. (1990), The Accumulation Buffer: Hardware Support for High-Quality Rendering, Siggraph 24(4): 309-318.

Heckbert, P.S., (1986), Survey of Texture Mapping, Computer Graphics and Applications 6(11): 56-67.

Heckbert, P.S. (1990), Adaptive Radiosity Textures for Bidirectional Ray Tracing, Computer Graphics 24(4): 145-154, Siggraph'90.

Heckbert, P.S. (1992), Discontinuity Meshing for Radiosity, Proceedings of the 3rd Eurographics Workshop on Rendering.

Ì ş ler, V., Aykanat, C., Özgüç, B. (1991), Subdivision of 3D Space Based on the Graph Partitioning for Parallel Ray Tracing, Proceedings 2nd EG Workshop on Rendering, Barcelona.

Jansen, F.W., Kok, A.J.F., Verelst, T. (1992), Hardware Challenges for Ray Tracing and Radiosity Algorithms. Proceedings 7th Workshop on Graphics Hardware, Cambridge England, September 1992, Eurographics Technical Report pp 123-134.

Kajiya, J.T. (1986), The Rendering Equation, Computer Graphics 20(4): 143-150, Siggraph'86.

Kedem, G., Ellis, J.L. (1984), The Ray Casting Machine, Proc. IEEE Int. Conf. on Computer Design: VLSI in Computers (ICCD'84), IEEE Computer Society Press, 533-538.

Kedem, G., Ellis, J.L. (1989), The Ray-Casting Machine. In: Dew, P.M., Earnshaw, R.A, Heywood, T.R., Parallel Processing for Computer Vision and Display, Addison-Wesley Publishing Company, p. 378-401.

Kirk, D., Voorhies, D. (1990), The Rendering Architecture of the DN10000VS, Computer Graphics 24(4): 299-307, Siggraph'90.

Kirk, D., Arvo, J. (1991), Unbiased Sampling Techniques for Image Synthesis, Computer Graphics 25(4): 153-156, Siggraph'91.

Kobayashi, H., Nishimura, S., Kubota, H., Nakamura, T. Shigei, Y. (1988), Load Balancing Strategies for a Parallel Ray-Tracing System Based on Constant Subdivision, The Visual Computer 4(): 197-209.

Kok, A.J.F., Jansen, F.W., Woodward, C. (1991), Efficient Complete Radiosity Ray Tracing Using a Shadow Coherence Method, Report of the Faculty of Technical Mathematics and Informatics, nr. 91-63, 1991. To appear in The Visual Computer.

Kok, A.J.F., Jansen, F.W. (1991), Source Selection for the Direct Lighting Computation in Global Illumination, Proceedings of the 2nd Eurographics Workshop on Rendering. To be published by Springer Verlag.

Kok, A.J.F., Jansen, F.W. (1992), Adaptive Sampling of Area Light Sources in Ray Tracing Including Diffuse Interreflection. Computer Graphics Forum 11(3): C289-C298, Eurographics'92.

Kok, A.J.F. (1993), Grouping of Patches in Progressive Radiosity, Proceedings 4th EG Workshop on Rendering (this volume).

Languénou, E., Bouatouch, K., Tellier, P. (1992), An Adaptive Discretization Method for Radiosity, Computer Graphics Forum 11(3): C205-C216, Eurographics'92.

Lee, M.E., Redner, A., Uselton, S.P. (1985), Statistically Optimized Sampling for Distributed Ray Tracing, Computer Graphics 19(3): 61-67, Siggraph'85.

Lischinski, D., Tampieri, F., Greenberg, D.P. (1992), A Discontinuity Meshing Algorithm for Accurate Radiosity, Computer Graphics and Applications 12(6):25-39.

Mitchell, D.P. (1987), Generating Antialiased Images at Low Sampling Densities, Computer Graphics 21(4): 65-72, Siggraph'87.

Nemoto, K., Omachi, T. (1986), An Adaptive Subdivision by Sliding Boundary Surfaces for Fast Ray Tracing, Proc. Graphics Interface'86, Computer Graphics Society, Montreal, pp. 43-48.

Nishita, T., Nakamae E. (1985), Continuous Tone Representation of Three-Dimensional Objects Taking Account of Shadows and Interreflection, Computer Graphics 19(3): 23-30, Siggraph'85.

Painter, J., Sloan, K. (1989), Antialiased Ray Tracing by Adaptive Progressive Refinement, Computer Graphics 23(3): 281-288, Siggraph'89.

Pattanaik, S.N., Mudur, S.P. (1992), Computation of Global Illumination by Monte Carlo Simulation of the Particle Model of Light, Proceedings 3rd Eurographics Workshop on Rendering, Bristol.

Pattanaik, S.N., Mudur, S.P. (1993), The Potential Equation and Importance in Illumination Computations, to appear in Computer Graphics Forum.

Priol, T., Bouatouch, K. (1989), Static Load Balancing for a Parallel Ray Tracing on a MIMD Hypercube, The Visual Computer 5(12): 109-119.

Pulleyblank, R.W. and Kapenga, J. (1986), VLSI Chip for Ray Tracing Bicubic Patches, In: Advances in Computer Graphics Hardware I, Springer Verlag, Proceedings First Eurographics Workshop on Hardware, 125-140.

Pulleyblank, R.W. and Kapenga, J. (1987), The Feasibility of a VLSI Chip for Ray Tracing Bicubic Patches, Computer Graphics and Applications 7(3): 33-44.

Rushmeier, H. (1988), Realistic Image Synthesis for Scenes with Radiatively Participating Media, PhD thesis, Cornell University, 1988.

Salmon, J., Goldsmith (1988), A Hypercube Ray-tracer. Proc. of the 3rd Conference on Hypercube Concurrent Computers and Applications Vol. II, ACM Press, pp. 1194-1206.

Scherson, I.D., Caspary, E. (1988), Multiprocessing for Ray Tracing: A Hierarchical Self-balancing Approach, The Visual Computer 4(4):188-196.

Shen, L.-S., Deprettere, E., Dewilde, P. (1990), A New Space Partition Technique to Support a Highly Pipelined Parallel Architecture for the Radiosity Method, In: Advances in Graphics Hardware V, Springer Verlag, Eurographics Hardware Workshop 1990.

Shen, L.-S., Laarakker, F.A.J., Deprettere, E. (1991), A New Space Partition Technique to Support a Highly Pipelined Parallel Architecture for the Radiosity Method II, In: Advances in Graphics Hardware VI, Springer Verlag, Eurographics Hardware Workshop 1991.

Shen, L.-S., Deprettere, E. (1992), A Parallel-Pipelined Multiprocessor System for the Radiosity Method, Proceedings of the Eurographics Hardware Workshop 1992. Eurographics Technical Report Series, p. 106-122.

Shirley, P. (1990) A Ray Tracing Method for Illumination Calculation in Diffuse Specular Scenes, Proceedings Computer Graphics Interface '90, p. 205-212.

Shirley, P., Wang, C. (1991), Direct Lighting Calculation by Monte Carlo Integration, Proceedings of the 2nd Eurographics Workshop on Rendering, Barcelona.

Shirley, P., Wang, C. (1992), Distributed ray tracing: theory and practice. Proceedings of the 3rd Eurographics Workshop on Rendering, Bristol.

Sillion, F., Puech, C. (1989), A General Two Pass Method Integrating Specular and Diffuse Reflection, Computer Graphics 23(3): 335-344, Siggraph '89.

Smits, B.E., Arvo, J.R., Salesin, D.H. (1992), An Importance-Driven Radiosity Algorithm, Computer Graphics 26(2): 273-282, Siggraph'92.

Speer, L.R., DeRose, T.D., Barsky, B.A. (1985), A Theoretical and Empirical Analysis of Coherent Ray Tracing, Proceedings Graphics Interface'85, Springer Verlag, pp 11-25.

Torborg, J.G. (1987), A Parallel Processor Architecture for Graphics Arithmetic Operations, Computer Graphics 21(4): 197-204, Siggraph'87.

Wallace, J.R., Cohen, M.F., Greenberg, D.P. (1987), A Two-Pass Solution to the Rendering Equation: A Synthesis of Ray Tracing and Radiosity Methods, Computer Graphics 21(4): 311-320, Siggraph'87.

Wallace, J.R., Elmquist, K.A., Haines E.A. (1989), A Ray Tracing Algorithm for Progressive Radiosity, Computer Graphics 23(2):315-324, Siggraph'89.

Ward, G.J., Rubinstein, F.M., Clear, R.D. (1988), A Ray Tracing Solution for Diffuse Interreflection, Computer Graphics 22(4): 85-92, Siggraph'88.

Ward, G.J. (1991), Adaptive Shadow Testing for Ray Tracing, Proceedings of the 2nd Eurographics Workshop on Rendering, Barcelona.

Whitted, T. (1980), An Improved Illumination Model for Shaded Display, Communications of the ACM 23(6): 343-349.

Xu, H., Peng, Q-S., Liang, Y-D. (1989), Accelerated Radiosity Method for Complex Environments, Proceedings Eurographics, Elsevier Science Publishers, p.51-59.

Yilmaz, A.C., Hagestein, C., Deprettere, E., Dewilde, P. (1989), A Hardware Solution to the Generalized Two-Pass Approach for Rendering of Artificial Scenes, In: Advances in Graphics Hardware IV, Springer Verlag, Proceedings Eurographics Hardware Workshop 1989, 65-79.