Tightly-Coupled Multiprocessing for a Global Illumination Algorithm

George Drettakis, Eugene Fiume and Alain Fournier †

Department of Computer Science, University of Toronto
10 King's College Road, Toronto, Canada M5S 1A4
{dret | elf} @ dgp.toronto.edu

A prevailing trend in computer graphics is the demand for increasingly realistic global illumination models and algorithms. Despite the fact that the computational power of uniprocessors is increasing, it is clear that much greater computational power is required to achieve satisfactory throughput. The obvious next step is to employ parallel processing. The advent of affordable, tightly-coupled multiprocessors makes such an approach widely available for the first time. We propose a tightly-coupled parallel decomposition of *FIAT*, a global illumination algorithm, based on space subdivision and power balancing, that we have recently developed. This algorithm is somewhat ambitious, and severely strains existing uniprocessor environments. We discuss techniques for reducing memory contention and maximising parallelism. We also present empirical data on the actual performance of our parallel solution. Since the model of parallel computation that we have employed is likely to persist for quite some time, our techniques are applicable to other algorithms based on space subdivision.

1. Introduction

In order to generate realistic images, global illumination algorithms tend to require vast amounts of main memory and computational resources. The demand for increasing visual realism is unlikely to diminish, and any progress made in improved visual realism simply results in increased expectations. Consequently, the performance of implementations of these algorithms on uniprocessors will continue to be unsatisfactory, despite constant improvements in CPU and memory speeds. Parallel solutions to such problems have often been advocated, but until the advent of affordable multi-processors, this option has not been widely available. These multi-processors are becoming quite popular, and it is evident that a popular mode of large-scale computation will be to have many such machines co-existing on a local-area network. Given this newly emerging computational environment, it is of some use and interest to apply it to intensive applications such as global illumination.

In this paper, we shall concern ourselves with a tightly-coupled parallel implementation of *FIAT*, a global illumination algorithm that we have recently developed [Four89]. The time and space requirements of FIAT often exceed the capabilities of large uniprocessors. However, it responds nicely to parallelism. While we have naturally tailored our parallel decomposition to the problem at hand, we feel that a number of more general principles arise from our work:

- that acceptable speedup requires careful reconsideration of the algorithm.
- that the architecture of the target environment must be taken into account in the parallel algorithm.
- that first-order approaches to parallelism such as simple locking and buffering achieve suboptimal and sometimes even subserial speedup.

† The authors gratefully acknowledge the financial support of the Natural Sciences and Engineering Research Council of Canada, the Information Technology Research Centre of Ontario, Apple Computer Inc., and the University of Toronto.

- that better serial algorithms can result from parallelism.

We believe that the basic tenets of our approach readily transfer to other intensive graphical applications.

In the next section, we give a brief overview of FIAT. Further details of the algorithm will be presented as needed. We shall discuss a specific parallel implementation of FIAT, and the effect of the implementation on the performance of FIAT. It is unfortunately not common for parallel approaches to progress beyond the paper-design stage. In our case, we are able to report on the performance of a true implementation. Lastly, we draw conclusions and examine further extensions to our work.

2. An Overview of FIAT

FIAT is a physically-based approach to the simulation of global (or environmental) illumination. Related approaches include radiosity [Gora84], ray-tracing [Whit80], and hybrid two-pass schemes ([Imme86],[Shao88],[Sill89]). Our approach differs in that it directly handles a wide class of light sources together with surfaces having arbitrary reflectance behaviours. This can be contrasted with traditional radiosity-based approaches which typically are constrained to diffuse reflectors and emitters, or ray-tracing approaches, which focus on specular reflectors and point or parallel light sources. FIAT directly handles the gamut of reflectance functions from the purely diffuse to the purely specular. A detailed presentation of FIAT is given in [Four89] and also in [Ouel89]. Our goal in this section is to describe very briefly the overall operation of FIAT, so that its parallel solution can be understood, and so that the application of our techniques to other algorithms is clear. FIAT is still somewhat experimental, but recent results have demonstrated novel illumination phenomena, such as secondary specular reflection and caustics - a phenomenon that is very difficult to achieve using other means (see Figure 1(a)).

FIAT takes as input a description containing geometric and light-source data. An 'initial cube' that bounds the scene is computed, and each face of the cube is tessellated into the same number of *side elements* or *sexels*. Each sexel occupies a small square on the side of a cell, and contains light information in the form of parallel beams of square cross-section. Each parallel beam represents a sample of light power emanating from the sexel in a specific direction. Many directions are used to sample a hemisphere of incoming light distribution. The goal of FIAT, coarsely speaking, is to convert global light information into a large set of sexels distributed throughout the environment. The sexels in turn contain many samples of light flux in various directions (beams). The initial cell, the sexels and the beams are all shown in Figure 2.

The reduction is achieved by maintaining a *power budget* for each cell, which takes into account incoming, outgoing, emitted, and absorbed light power. A cell is *balanced* if the incoming and emitted light power totals the outgoing and absorbed power. There is initially just one cell, the bounding volume, and the process begins by computing the power entering the volume from external light sources. This results in a net power *imbalance*, which FIAT will try to rectify. If a cell is empty or contains objects that can be dealt with directly, then a new power balance is determined. If the cell cannot be dealt with directly, then it is subdivided in octree fashion, and relevant sexel information is transferred to the subcells.

We consider in slightly greater detail the case in which a cell can be dealt with. If a cell is empty, we simply propagate every incoming light beam to relevant outgoing sexel. If the cell is not empty, an intersection computation between a specific light beam and the scene within the cell is performed. Light beams that do not intersect with any objects within the cell are propagated as in the empty-cell case. For those beams that do intersect with an object, a new power distribution about the point of intersection is computed by invoking a local illumination model, with the beam acting as a light source. The outgoing distribution may be quite complex, since it is not assumed that the reflectance function of the intersected object is perfectly diffuse or specular. This distribution is sampled in many directions, and the relevant outgoing sexels are updated. If the intersection point corresponds to a point that is visible from the eye position, then the shade of this point in the direction of the viewpoint is computed and added to a relevant pixel entry in a *light update buffer* or *LUB*. Thus an image is progressively produced as a function of cell

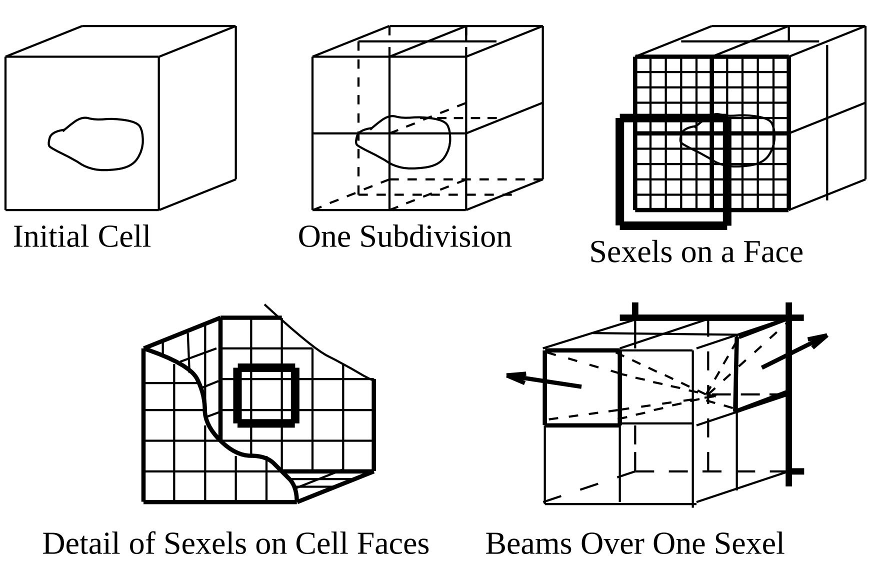


Figure 2: Initial Cell, Subdivision, Sexels, Beams

balancing. The algorithm continues until the overall power imbalance falls below a threshold value.

3. The Goals for a Tightly-Coupled Parallel Solution

The task of FIAT inside a cell is complex and can involve a large amount of computation and data. To take maximum advantage of the capabilities of parallel hardware we set the following goals:

(a) To perform as much of the computation as possible in parallel.

(b) To distribute the computation evenly among all processors.

(c) To find a solution that minimises the amount of locking necessary, and allows the processors to compute independently as much as possible.

(d) To introduce a solution that can be scaled to a large number of processors. The range of processors in which we are interested is that of existing common-bus shared-memory architectures, incorporating from 2 to 16 processors and not more than about 30. For larger-scale parallelism, we prefer to take a hierarchical approach in which several tightly-coupled processors are employed. This approach is an orthogonal dimension to the techniques described in this paper [Dret90].

To achieve these goals we shall first analyse the serial algorithm to determine the computationally expensive sections. We investigate the structure of the algorithm and the data dependencies involved, to decide how much computation in these sections can be parallelised, thus achieving goal (a). Goal (b) is achieved by investigating the ways in which the computation can be segmented into independent components so that the work is evenly distributed. To avoid locking it may be necessary to use temporary memory private to each processor, and examine whether such a solution is actually preferable. For both (b) and (c) in combination with private buffering schemes, changing the order of computation proves to be necessary. Finally the selection of the scheme will be such that goal (d) is satisfied by design.

In the remainder of the paper we present the serial algorithm in more detail, outline a set of test cases which we use to analyse the behaviour of the algorithm, and investigate the choices of parallel schemes. We present one solution that satisfies the goals to the largest extent, and discuss some of the problems that were dealt with in the design and implementation of this solution. Timing results on three test scenes are presented showing the effect of varying several parameters affecting performance. Some possible alternative schemes for parallelism are discussed, and we close with a summary and some possible enhancements.

4. Analysis of the Serial Algorithm

The computation in a cell can be split into two components. The first, called *balancing*, involves distributing light entering a cell as a result of interaction with objects in the cell; also involved is the propagation of light beams within the sexels on one face of a cell to the sexels on some other face if the beam does not intersect with anything. The second component, *shading*, involves updating the section of the LUB associated with the cell. This is done by determining the visible surface for each pixel and shading based on the incoming light that impinges on this visible point. These two processes are independent, but use some common subroutines.

4.1. The serial algorithm.

For the purposes of this discussion we present a high-level pseudo-code version of the balancing (Figure 3(a)) and shading algorithms (Figure 3(b)) for the serial version of FIAT, shown on the following page.

4.2. Analysing the behaviour of FIAT in a cell.

From the pseudo-code presented in Figure 3 it is evident that a large proportion of the computation will be spent in the functions that are executed in the inner loops. For balancing, these are the beam intersections and, if distribution is needed, the power calculations and the code that finds the destination sexel. For shading, the corresponding expensive operations are finding the sexels that shine on the visible point, and performing power calculations to determine the shade. To confirm these intuitive estimations profiling was performed for test cases. These test cases will also be used to evaluate the performance of the parallel solution proposed.

```
BalanceCell (cell)
{
    for each face
        for each sexel
            for each beam
                /* does the beam hit anything ? */
                P = BeamIntersect
                if something is intersected at point P
                    DistribLight(P)
                else
                    PropagateLight(beam)
}

DistribLight(hit_point)
{
    for every direction d in sphere of directions around hit_point
        /* calculate power reflected/refracted in direction */
        Power = PowerCalc()
        Sexel = FindDestinationSexel()
        assign Power to direction d on Sexel
}

PropagateLight(beam)
{
    sexel = FindDestinationSexel
    assign the power of beam to same direction on sexel
}
```

Figure 3 (a) The serial algorithm for balancing.

```
ShadeCell (cell)
{
    determine xmin,xmax ymin,ymax corresponding to
    projection of cell onto LUB
    for x = xmin to xmax
        for y = ymin to ymax
            /* intersect ray from eye through centre of pixel (x,y) */
            P = RayIntersect()
            if P is on an object inside cell
                /* find appropriate sexels whose
                   beams impinge on P */
                while (sexel = FindHitSexel())
                    /* calculate power that sexel shines on P and shades (x,y)  */
                    ShineBeam(sexel, P, x, y)
}
```

Figure 3(b) The serial algorithm for shading.

4.2.1. Test Cases

FIAT is extremely flexible in the range of light sources and surface properties that can be simulated. Test cases must therefore be chosen with care to exercise as many modalities as possible. We have employed the following three test scenes.

Scene 1: A shiny ashtray (60% specular, 40% diffuse) lit by a directional light source that is at a 45 degree angle. The model consists of 600 triangles, and as a result many cells contain more than one object. We employ directional light sources to illuminate this environment, which results in the creation of caustics from concentration of light due to reflection. Figure 1(a) shows this scene computed at higher subdivision to demonstrate the effect more clearly.

Scene 2: A room that has 3 walls, a red left wall, a blue right wall, a grey back wall, and a grey ceiling and floor. There are 12 triangles in this scene. All surfaces in this scenes are mostly diffuse, and absorb some of light falling on them. The scene is lit by a extended area light source that is where a front wall would be if the room was closed all around. This scene shows colour bleeding, similar to that shown by radiosity on the ceiling from the two side walls. Figure 1(b) shows this scene computed using the reconstruction techniques that have been developed for FIAT, eliminating some of the artifacts.

Scene 3: The same as Scene 2, with the difference that the floor is now shiny (60% specular 40% diffuse) and acts as a mirror. The image is similar to that presented in Figure 1(b), with a different floor.

For scenes 2 and 3 only a small number of sexels and cell subdivisions are employed. Because memory requirements for these scenes can exceed real memory on some machines, it is occasionally necessary for the purposes of analysis to use scene subsets. For scenes 2 and 3 the subset scenes have the floor and the back wall removed, and for scene 3 the ceiling is shiny instead of the floor that is no longer present. We refer to these scenes as scenes 2a and 3a respectively.

4.2.2. Profiling results.

By using standard profiling tools we have determined which functions for each corresponding part of the balancing and shading computations are the most expensive, and are therefore prime candidates for parallelism. The profiling results confirm what was expected from examining the structure of the algorithm. These results show that overall more time is spent balancing than shading. In all cases 50-60% of the execution time is spent in `BalanceCell`. Of this time, for Scene 1, about 40% is spent in applying the local illumination model at the intersection point to determine the power outgoing in each direction (`PowerCalc` and power assignments). The additional 10% is spent determining which sexel is affected. For Scenes 2 and 3 the time is split almost equally between power calculations and determining the destination sexel. This discrepancy is due to the nature of the ashtray surface, that is highly specular, thus reflecting light in a limited number of directions. The beam intersection operation is also somewhat expensive (2-3% of the computation time).

For shading the time is almost equally split between finding the sexel that affects the visible point being considered (`FindHitSexel`) and the time spent applying the local illumination model to shade the point (`ShineBeam`). This is true for all scenes.

4.3. Linearity of Light Interaction

In order to allow progressive updates to the light update buffer (LUB) that do not depend on the order of application, we have assumed that light-object interaction is a linear operation [Four89]. For most materials and light sources, this is a perfectly valid assumption. A consequence of this assumption is that light beams can be propagated or distributed in any order, yielding the same cumulative result. This is very important since it facilitates the reorganisation of a large amount of the computation in ways that are more convenient for parallelism.

5. A Parallel Decomposition Strategy

To devise a good parallel decomposition strategy, one must identify (potentially) non-interfering portions of the computation. It is often the case that although independent tasks exist, parallel execution can result in corruption of some data structures that tasks access at the same time. The obvious first-order approach is to require that a task acquire a lock before entering the critical region of code that modifies common data. The gain in parallelism in this case is often suboptimal. Another solution involving more intensive analysis is to reorganise the computation such that tasks no longer write the same memory locations, either by using a private buffer in each processor or by eliminating the need for such a write. Finally a scheme must be developed that actually assigns these parallel tasks to processors in a manner that keeps each processor as busy as possible at all times. In discussing possible parallel strategies we will address balancing and shading since they are independent computations.

5.1. Parallel Balancing.

For each incoming beam in a sexel we have to perform an intersection with the objects in the cell. The intersection operation of an incoming beam is independent of any other intersection of the same form, since no data is written apart from a temporary structure that contains the hit point information if such a point exists. Thus, these intersections can be easily performed in parallel.

Most of the operations involved in the distribution of a light beam about an intersection point are independent. These operations include invoking the local illumination model, finding the sexels to update, and power computations. The complicating issue is that more than one intersection point may contribute to the same outgoing direction of the same sexel (see Figure 4), since there is only one representative of each direction for each sexel.

A more subtle problem appears if we choose to parallelise the actual distribution of light around an intersection point. In `DistribLight` (Figure 2(a)) we see that at every intersection point a power calculation occurs for each discrete direction in the sphere of directions placed on that point. An obvious way to parallelise is to perform the power calculations for each direction in parallel. At a first glance this seems a reasonable strategy. However FIAT requires that the local illumination model be normalised such that the total power leaving an intersection point is no larger than the power entering it. In other words if the power impinging at point P from a given direction was P_{imp}, and the model used assigns P_{out_i} power to be outgoing in direction i (Figure 5), we wish the total outgoing power P_{out} (if d is the number of directions in the sphere around P):

$$P_{out} = \sum_{i=0}^{d} P_{out_i} \leq P_{imp}.$$

In any physically-based local illumination model, power must be conserved. Unfortunately, few models in computer graphics have this property, and so a pre-normalisation step is required. In the initial serial implementation of FIAT, each P_{out_i} is multiplied by a scaling factor S defined as follows.

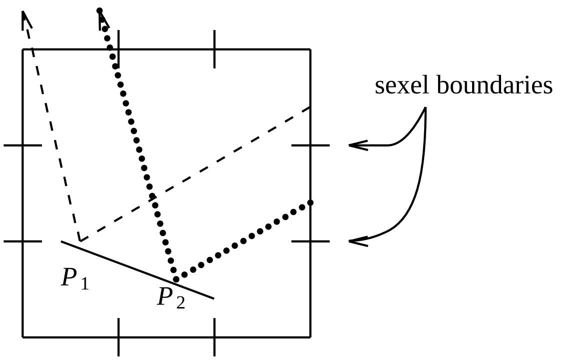


Figure 4: Different Intersection Points Update Same Beam.

$$S = \frac{P_{imp}}{\sum_{i=0}^{d} P_{out_i}}. \tag{1}$$

Consequently,

$$P_{out} = \sum_{i=0}^{d} S\, P_{out_i} = S \sum_{i=0}^{d} P_{out_i} = \frac{P_{imp}}{\sum_{i=0}^{d} P_{out_i}} \sum_{i=0}^{d} P_{out_i} = P_{imp} .$$

Normalisation guarantees conservation of power even for unrealistic illumination models. The final value assigned to the destination sexel is $S\, P_{out_i}$.

In the serial case, the value S can be computed on-the-fly, since one processor handles all the P_{out_i}. This is clearly not the case in a parallel solution, so one must arrange to precompute a representative S. Otherwise some processors may have to wait for others to complete before being able to apply S to their outgoing power values. Later, we shall see how a good value for S can be precomputed, so that this potential barrier is removed entirely. We stress that this problem only arises when local illumination models that do not conserve power are employed.

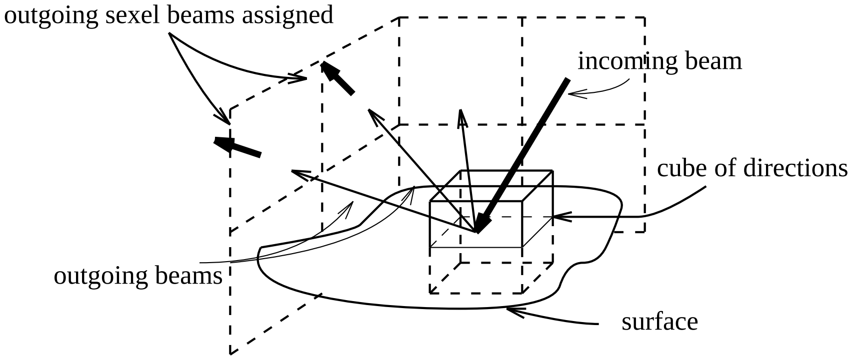


Figure 5: Outgoing power in direction *i*

5.2. First-Order Parallel Balancing

Without changing the structure of the initial algorithm, we can statically divide the processing of faces, sexels or beams among processors. In such a scheme the main loop of `BalanceCell()` is executed on every processor but only certain of the faces, sexels or beams are dealt with by each. Parallelism at the face or sexel level is unsuitable for two main reasons. First, such an approach does not scale well (there are 6 faces, and in the worst case there are 6 sexels in a cell). Second, it does not provide very good load balancing as there may be one face or only a small number of sexels that contain all the incoming light. At the beam (direction) level, things are much better as there are usually many beams (a typical instance of FIAT has 192 such beams per sexel), and they are usually evenly distributed. However such a scheme may suffer from the contention problem described in the previous section and illustrated in Figure 4. We briefly describe three ways of addressing this problem without significantly altering the overall structure of the initial serial algorithm.

Locks. Locking critical sections is straight-forward, but can cause excessive waiting. Moreover, the use of many locks (as would be the case in FIAT if each sexel were to have a lock) would pose high additional memory requirements.

Private buffers: Perform all sexel updates in a private buffer, acquire a lock when the buffer is full, and write the updates to the real sexels. This also suffers from the problem of potentially long delay on locks as the lock must be acquired for the full duration of the buffer write-back. A severe practical drawback is that in a memory-intensive application like FIAT, real sexel information migrates to distant locations in memory while buffers are filling. In systems that use look-aside buffers for memory accesses, processor faults are caused at an excessive rate every time the buffer has to be written back (typically thousands of times a second). When this problem appears, the actual performance of this approach can even be sub-serial.

Pooled Buffers. A third solution is to employ a two-pass approach. Each sexel is preassigned to a processor in some static manner. If there are n processors, each processor maintains n buffers. In the first pass each processor propagates and distributes light, and when a sexel must be updated, it inserts the update information in the buffer corresponding to the processor assigned to this sexel. In the second pass each processor runs through the buffers containing updates for the sexel assigned to it on all the processors, performing the real updates. This solution avoids the problem of frequently accessing distant memory as the set of sexels that each processor accesses is smaller; during the first pass only local accesses are required. This solution yields relatively good results but is limited by the buffer-management and synchronisation overhead.

From the above discussion we see that simple solutions that do not alter the original structure of the serial algorithm or have marginal changes are limited due to synchronisation, memory access or locking constraints. Shortly, we shall describe a solution that allows these problems to be circumvented so that satisfactory speedup is achieved.

5.3. Parallel Shading

Shading presents much less difficulty than balancing. The reason for this is that the only data being written in the shading process are the LUB pixels. Therefore if a scheme is found that assigns each pixel to a processor, there is no contention, and each processor can process the pixels assigned to it independently without requiring any synchronisation or locking.

6. A Fast Low-Contention Parallel Balancing Scheme

To obtain maximal parallelism and minimal memory contention, reorganisation of the computation from the initial serial algorithm is required. Both propagation and distribution of light around an intersection point are based on a global sphere of directions. In the case of propagation, the set of directions within a sexel are defined such that a beam originating from one sexel always updates the beam of a sexel on a different face that has the same direction as the source beam (see Figure 6(a)). In the case of distribution around an intersection point P, the power outgoing in direction i is assigned to a sexel (or sexels) in direction i (Figure 6(b)). These two observations lead to a natural task decomposition scheme for parallelism:

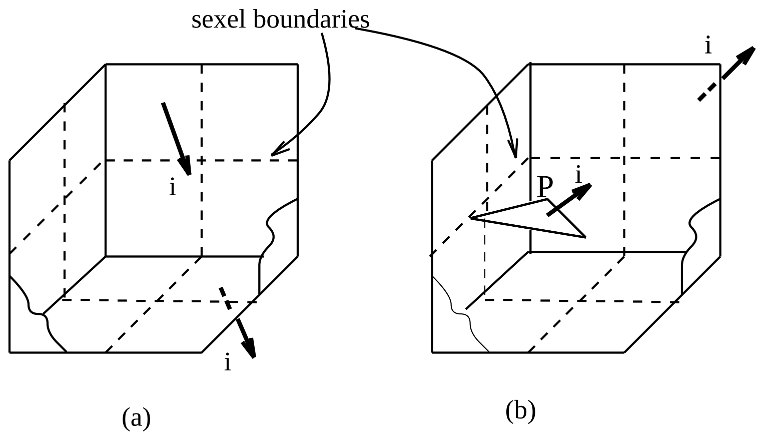


Figure 6: Assigning directions.

Each direction is assigned to a different processor, making this processor responsible for all updates in this direction on any sexel, as a result of either propagation or distribution.

For propagation, this scheme is simple, since the serial counterpart can remain intact with each processor simply working only on those beams that are assigned to it. For distribution of light around intersection points it is necessary for the all intersection points to be available to all processors. In this way each processor can perform the computation for the directions assigned to it for all intersection points.

To allow this, a two-phase buffering scheme is used. In the first phase each processor performs the intersections of the beams it is responsible for, saving the intersections in a private buffer residing in shared memory. When the buffers fill, the processors synchronise and the second phase begins. At this point, each processor runs through the buffers corresponding to all the processors and for every intersection performs the power calculations only for the directions assigned to it and updates these directions on the relevant sexels.

As described earlier, we must ensure that the local illumination model is properly normalised before distribution, for otherwise all processors would have to synchronise at every intersection point to use the total outgoing power for the scaling factor. This would create unnecessary waiting and would severely degrade performance. Fortunately, a good candidate for a normalisation constant is at hand. We shall derive an upper bound for the amount of power that can possibly leave a point P on a given surface if incoming power P_{imp} impinges on P. This upper bound was developed for other purposes in [Ouel89], but however is exactly what is necessary to overcome this problem. Assuming a Blinn-like reflection model for a surface with constant $n > 0$, an upper bound P_u on outgoing power is:

$$P_u = \int_0^{2\pi} \int_0^{\pi/2} \cos^n x \; dx \; dy = 2\pi \int_0^{\pi/2} \cos^n x \; dx.$$

We can use P_u in place of $\sum_{i=0}^{d} P_{out_i}$ in equation (1), giving us a revised value of S:

$$S = \frac{P_{imp}}{P_u}. \qquad (2)$$

P_u can be precomputed once for each distinct n using a numerical integration package, or an analytical derivation if n is integral, and stored with the surface description. To perform normalisation each process needs only to apply equation (2), and synchronisation is avoided.

We summarise our approach by the following pseudo-code. On each processor the code shown in Figure 7 is executed.

```
ParallelBalanceCell()
{
    for each face
        for each sexel
            for beams in directions assigned to this processor
                P = BeamIntersect
                if something is intersected at point P {
                    BufferIntersection(P)
                    if ( BufferFull )
                       BufDistribute();
                }
                else
                    PropagateLight(beam)
}

BufDistribute()
{
    for each processor's buffer
       for each intersection in buffer
          for each direction d assigned to this processor
            calculate power reflected/refracted in direction d
            normalise using Pu
            assign power to beam in corresponding sexel
}
```

Figure 7: Parallel Balancing Scheme.

6.1. Goals Achieved by the Parallel Approach

The solution just described achieves the goals set out at the beginning of this paper:

- All expensive portions of the computation are performed in parallel (i.e. beam intersections, power calculations, destination sexel determination for distributing).
- The scheme has a satisfactory 'natural load balancing' property, because load distribution is tied to distribution of directions.
- Memory contention is minimised and locks are almost entirely avoided. This solution also has the benefit that there are very few barriers at which the processors have to synchronise. The only required synchronisation point occurs when the intersection buffer fills.
- The solution can in principle be scaled to the number of processors equal to the number of directions. However the benefit of this order of parallelism would quickly be overcome by the overhead of maintaining and synchronising a large number of processors. Although we report results on machines consisting of 4 processors or fewer, it seems that any number of processors that fit on a single-bus system (less than 32 in most existing systems) would perform well.

From the above we see that from the point of view of design this scheme has the desirable properties that are required from the parallel balancing scheme. As we shall see in the section on timing results the solution performs satisfactorily on a real parallel system.

7. A Parallel Approach to Shading

From Figure 3(b) we can see that parallelising the shading computation is an easy task. All that is required is to assign each pixel to a different processor that then performs the ray intersection and the power calculations when collecting light from the relevant sexels in parallel. If we have n processors, the parallel shading algorithm executed on processor i in parallel is shown in Figure 8. For every scan line each processor with processor-id i shades pixels $i + jn$ where n is the number of processors and $a \le j \le b$ such that $a = \min k : xmin \le (i + kn)$ and $b = \max k : (i + kn) \le xmax$.

The shading scheme successfully achieves the goals put forward:

- Shading expense is determined by light collection, which requires a power calculation for the light shining from the relevant sexels, finding these sexels and in the intersection of the ray from the eye. All are performed in parallel.
- The load balancing is dependent on the values of `xmin, xmax, ymin, ymax`.
- This scheme requires no locking or synchronisation for the duration of the shading computation.
- The parallel shading approach we have proposed scales theoretically to as many processors as there are pixels in a region of the LUB related to a cell, although diminishing returns would likely set in well before this upper bound is reached.

```
/* shading code */
ParShadeCell ( i)
processor-id i
{
    for y = ymin to ymax
       x = xmin + i
       while ( x < xmax ) {
            intersect ray from eye through centre of pixel (x,y)
            if object hit inside cell
                collect light from all relevant sexel beams
                shade pixel (x,y)
            x += n
      }
}
```

Figure 8: Parallel Shading Scheme.

8. Effect of the Parallel Solution on the Serial Algorithm

As we tested the parallel implementation, we discovered an interesting result. The parallel algorithm was tested initially for various test cases that fit in the real memory of the systems used. As soon as subdivision levels in FIAT are raised the memory used quickly becomes much larger than real memory, therefore incurring the overhead of paging. Surprisingly, comparisons with the original algorithm appeared to be giving the parallel version speedup 2-2.5 times the number of processors. For example, on a 4 processor system the speedup was 9.8. It quickly became clear that the reorganisation of the computation by buffering intersection points and separating the direction space into sections equal to the number of processors provided better memory-access patterns. By simulating the parallelism in the uniprocessor case the serial version achieved much better performance when paging overwhelms the computation. Thus our parallel algorithm improved serial performance as well.

9. Timing Results

Tables 1-5 show the timing results for the three selected test scenes. Also shown is speedup and processor utilisation (efficiency), and scaled speedup and scaled processor utilisation. Let t_n be the running time of the implementation on $n \ge 1$ processors. We define the speedup $S_n = t_n / t_1$, and the processor utilisation $PU_n = S_n / n$. The serial time reported in the results is the time in which the parallel algorithm must run in serial on one processor, and includes things such as the time spent reading in the scene, which is obviously large for the 600 triangles in scene 1. The serial time is the average of the reported times spent in serial mode for each run that differed by a maximum of .5 seconds. If s is this serial time we define scaled speedup as $S'_n = (t_n - s) / (t_1 - s)$ and scaled processor utilisation is $PU'_n = S'_n / n$.

Tables 1-3 were computed on an IRIS 4D/240 with 4 16MHz MIPS R3000 processors and 64 Megabytes of real memory, while the results reported in Tables 4 and 5 were computed on an IRIS 4D/120S with 4 10MHz MIPS R2000 processors and 16 Megabytes of main memory. The implementation is in ‘C’ using the system-provided parallel support library routines for parallel tasks and semaphores.

Table 1: Timing results for Scene 1

# Procs	Tot. Time	Ser. Time	S	PU	S’	PU’	Size (MB)
1	15:28.7	-	-	-	-	-	-
2	9:09.4	35.0	1.69	84%	1.73	86.5%	29.0
3	7:20.9	35.0	2.10	70%	2.20	73.3%	29.2
4	6:22.4	35.0	2.43	61%	2.57	64.3%	29.3

Table 2: Timing results for Scene 2

# Procs	Tot. Time	Ser. Time	S	PU	S’	PU’	Size (MB)
1	47:39.0	-	-	-	-	-	-
2	26:38.5	27.3	1.78	89%	1.80	90.0%	44.7
3	19:10.3	27.3	2.4	80%	2.52	84.0%	44.1
4	16:09.0	27.3	2.95	73%	3.0	75.0%	44.5

Table 3: Timing results for Scene 3

# Procs	Tot. Time	Ser. Time	S	PU	S’	PU’	Size (MB)
1	1:09:11.0	-	-	-	-	-	-
2	37:53.2	28.5	1.82	91.0%	1.84	92.0%	47.6
3	28:21.6	28.5	2.43	81.3%	2.46	82.0%	48.1
4	22:53.4	28.5	3.02	75.5%	3.06	76.5%	47.7

Table 4: Timing results for Scene 2a

# Procs	Tot. Time	Ser. Time	S	PU	S’	PU’	Size (MB)
1	13:11.8	-	-	-	-	-	-
2	6:55.1	12.5	1.91	95.5%	1.93	96.0%	14.7
3	4:53.7	12.5	2.69	89.6%	2.86	95.3%	14.8
4	3:52.7	12.5	3.40	85.0%	3.54	88.5%	14.9

Table 5: Timing results for Scene 3a

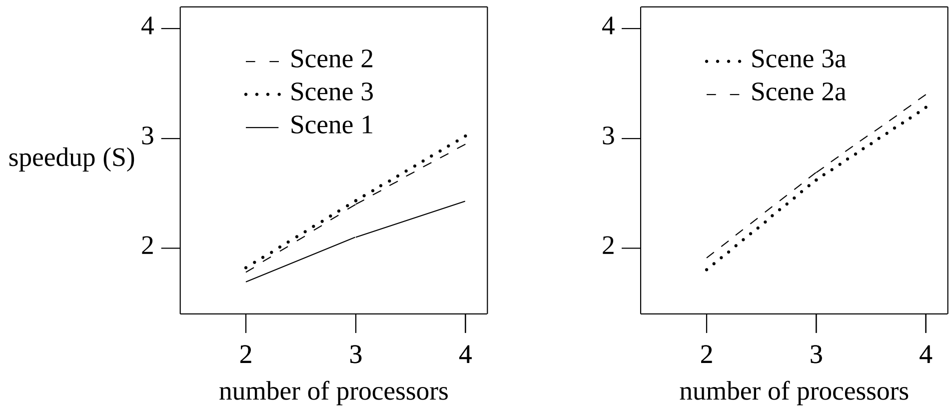
# Procs	Tot. Time	Ser. Time	S	PU	S’	PU’	Size (MB)
1	12:58.7	-	-	-	-	-	-
2	7:10.6	12.6	1.80	90.0%	1.83	91.5%	14.8
3	4:57.0	12.6	2.62	87.0%	2.69	89.6%	14.8
4	3:56.9	12.6	3.28	82.0%	3.41	85.2%	14.9

The speedup and efficiency achieved are generally satisfactory. There are two factors affecting speedup that can be noted from the results.

Light source and surface types. Our solution depends on the type of light sources and the type of surfaces existing in the scene. The results for scene 1 are worse overall, since the light source is directional and the surfaces are specular. This means that a limited number of directions have more power than others and as a result processor utilisation is suboptimal.

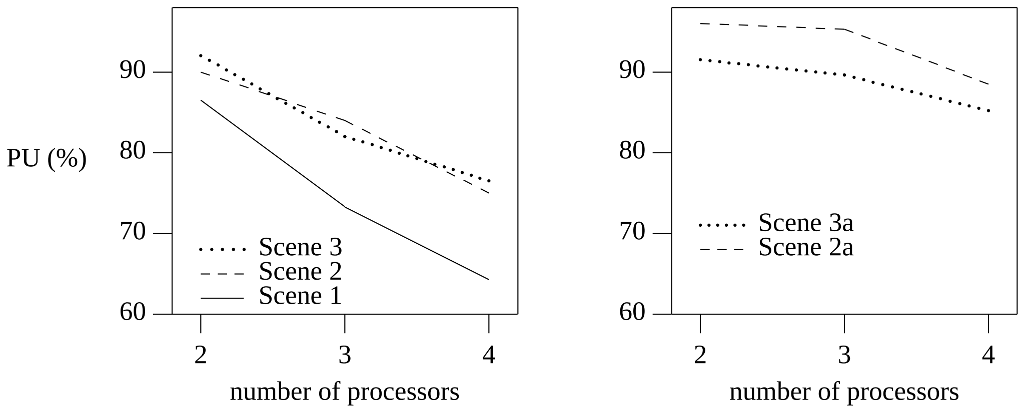
Large memory requirements and hardware. In scenes 1-3 we see worse performance than scenes 2a and 3a. The reason for this is the memory system hardware of the specific systems used. The special fast look-aside buffers are strained with the memory requirements of FIAT in each cell, and since there is only one memory servicing all the requests there is an effect of serialisation when memory requirements are acute. This fact is confirmed using the system monitoring facilities that report look-aside buffer faults in the range of 3000-4000 faults/second. This is one of the problems that affects the degradation of efficiency as the number of processors grow.

The following graphs present speedup by the number of processors from 2 to 4. Graph 1 shows the speedup for the scenes 1, 2 and 3 and graph 2 for scenes 2a and 3a. We see from these that the speedup for the small scenes (2a, 3a) is better, and there is little degradation as the number of processors grow.



Graphs 1 and 2: Speedup for Scenes 1, 2 and 3 and for Scenes 2a and 3a.

The following two graphs show processor utilisation for the two sets of scenes, again scenes 1, 2 and 3 that have larger memory requirements (Graph 3), and scenes 2a and 3a in Graph 4. From these graphs, which use the same scale for purpose of comparison, observe that the penalty for each extra processor for the large memory cases is 6-12% while for the scenes with low memory requirements it is much less varying from 2-5%. This means that if the memory requirements are smaller the solution can scale to a larger number of processors.



Graphs 3 and 4: Processor utilisation for Scenes 1, 2 and 3 and for Scenes 2a, 3a.

10. Conclusions and Future Directions

We have presented an approach that allows the use of parallelism for FIAT, a complex global illumination space-subdivision algorithm, and have successfully demonstrated significant speed-up with the addition of processors. The techniques readily migrate to other parallel environments: we have recently migrated the parallel implementation of FIAT to a 7-processor DEC Firefly. We also believe that our techniques to reduce memory contention and locking can be employed in other applications.

FIAT has evolved since the approach described in this paper was originally developed. Reconstruction techniques are used in both the propagation and distribution phase and in the shading, to allow a better representation of light flux using the discrete samples. The methods described in this paper are directly applicable to the new technique, with one exception: there is no longer a one-to-one correspondence between the originating sample (direction) and the destination sample, since more than one sexel and more than one direction are affected. This problem can be overcome with some additional effort.

In [Dret90] a full hierarchical solution is designed and implemented. The octree structure used to represent the cell subdivision is distributed across a network of parallel processor stations. The tightly-coupled solution described here is run on each such station, and special algorithms and techniques are developed to allow maximal benefit by distributing both memory and computation.

References

[Dret90] Drettakis, George, "Hierarchical Parallelism for a Global Illumination Algorithm", *M.Sc Thesis, Department of Computer Science, University of Toronto*, 1990.

[Four89] Fournier, Alain F., Eugene L. Fiume, Marc J. Ouellette and Chuan K. Chee "FIAT LUX", *D.G.P. Technical Memo.*

[Gora84] Goral, Cindy M., Kenneth E. Torrance, Donald P.Greenberg and Bennet Battaile, "Modelling The Interaction Of Light Between Diffuse Surfaces", *ACM Computer Graphics 18(3),* July 1984

[Imme86] Immel, David S., and Michael F. Cohen, "A Radiosity Method For Non Diffuse Environments", *ACM Computer Graphics* 20(4), August 1986.

[Ouel89] Ouellette, Marc, "Modelling Light Propagation and Global Illumination", *M.Sc. Thesis, Department of Computer Science, University of Toronto*, 1989.

[Shao88] Shao, Min-Zhi, Qun-Sheng Peng and You-dong Liang, "A New Radiosity Approach by Procedural Refinements for Realistic Image Synthesis". *ACM Computer Graphics* 22(4), August 1988.

[Sill89] Sillion, Francois and Claude Puech, "A General Two-Pass Method Integrating Specular and Diffuse Reflection", *ACM Computer Graphics* 23(3), August 1989.

[Whit80] Whitted, Turner, "An Improved Illumination Model for Shaded Display", *Comm. of the ACM*, Vol 23, no 6, June 1980.

Figures 1(a)-(b). Figure 1(a) depicts Scene 1 calculated at high subdivision, showing a caustic in the centre of the ashtray. Figure 1(b) depicts Scene 2, calculated using the new reconstruction techniques for both distribution and shading. Colour bleeding is clearly shown.