Parallel Subpixel Scanconversion

Ute Claussen

Eberhard-Karls Universität Tübingen
Wilhelm-Schickard-Institut für Informatik
Graphisch-Interaktive Systeme

1. Introduction

Computer graphics and its subsections image processing, image analysis and image generation are known to be a wide field for the application of parallel architectures. While in image processing and analysis the demand for "real time" computation is in the center of discussion, it becomes more and more important in the field of image generation, too. Some applications, like sequences of realistic appearing images raise highest demands on algorithm and architecture as well.

We want to tackle the problem of image generation on a raster device. This computation can be subdivided into the geometric processing (including transformations and clipping) and the pixel processing (including scanconversion, visibility computations, anti-aliasing, and shading). You can see that in the first phase this is an *object oriented* computation and in the second phase it's a *pixel oriented* computation (figure 1). Both in the first and second phase parallel processing is possible.

The attempts to the parallelization can be classified according to this subdivision. There are object oriented, pixel oriented (or more generally: image space oriented) and hybrid solutions. As a result, the boundary between pixel oriented and object oriented processing is often pushed to the periphery, including drawbacks in the computational speed.

The general approach to solve the parallelization problem is shown in figure 2. Because of the increasing overhead in partitioning the whole problem two times the extreme proposals partitioned exclusively object space or exclusively image space. This consequently leads to architectures including one processor per object or one processor per pixel (e.g. the pixel-planes of Fuchs et. al. [FuPo-81] and [Fuea-85]).

But both attempts are showing drawbacks: a strict distribution of objects results in bottlenecks in the hardware, strictly distributing pixels results in aliasing effects.

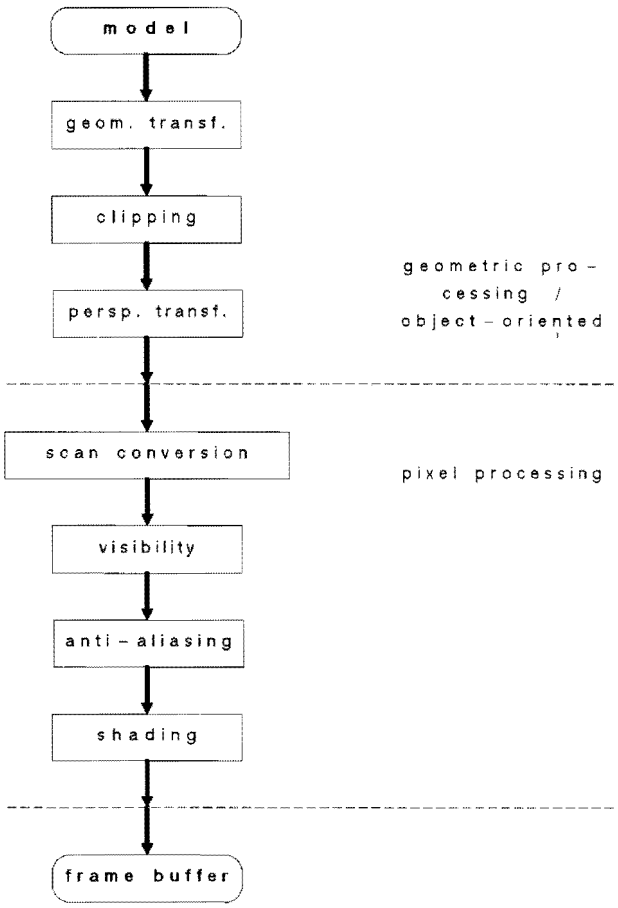

Figure 1: Image Generation Pipeline

An alternative proposal was made by Strasser last year ([Stra-85], [Stra-86], and [Wein-81]). He concentrated only on the pixel processing. Like Weinberg he proposes an object processor pipeline, which stores the object information and pipelines the pixel information. Figure 3 shows his concept.

With this proposition the possibility to partition the image generation in such a way that a) real-time generation is feasible and b) the architecture can be implemented by VLSI-design is proved. We want to use this architecture as a basis for further investigations.

Though the proposition shows some drawbacks such as the application dependence and some structure dependent delays, we want to show in this paper that the integration of subpixel scanconversion without loss of the real time feature is possible. Application dependence is due to the use of one processor per object. A fixation of the number of processors leads directly to a limitation of the complexity

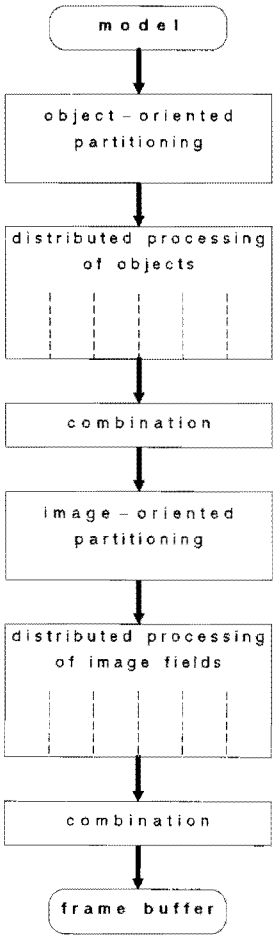

Figure 2: General Approach to the Parallelization of Image Generation

of the scene (e.g. number of objects in the scene). Furthermore the use of planar triangulated polygons can be seen as a drawback. But that fact can be overcome by changing the functionality of the object processors.

Tolerance can be shown towards the fact that the pipeline must be loaded sequentially with the object information. Parallel proceeding is desirable because only a constant time per frame would be necessary. On the other side it has been demonstrated that first it isn't possible to do that by todays technology and second it is not necessary, because the time constraints can be met without it !

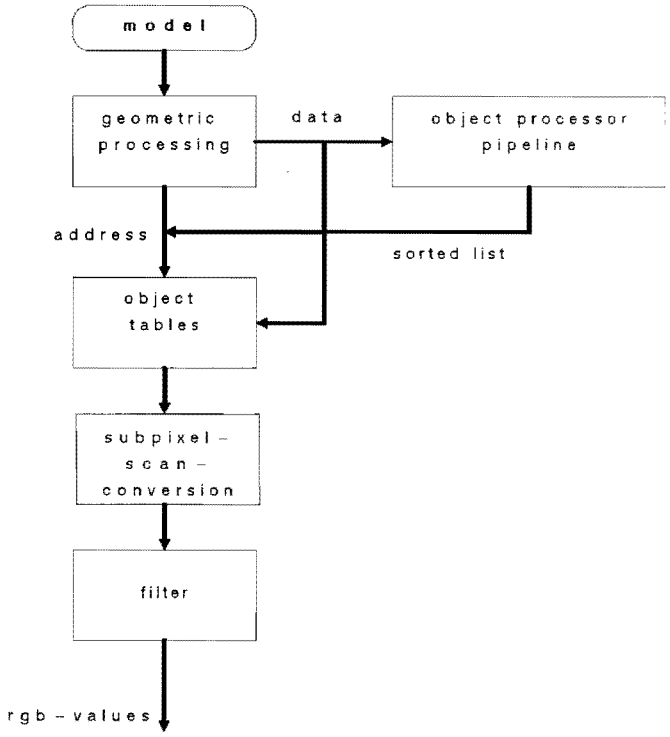

Figure 3: System Architecture

2. Subpixel Scanconversion

Further points that were performed insufficiently in the proposed architecture are the subpixel scanconversion and the shading of the polygons. These two steps are necessary to complete the pipeline presented in figure 1. In this article we will concentrate on the subpixel scanconversion.

Subpixel computations are known as a solution for the well known problem of "staircases" which appear on raster displays (see [FiFR-83] for a proposition of algorithm and architecture). This method subdivides a pixel into a field of $M' \times M'$ subpixel on which the same algorithms are applied. Then the field of subpixel values is used to determine the pixel value, generally by weigthed filtering.

So, if the problem of scanconversion is:

scanning of N objects on a $M_1 \times M_2$ pixel raster display,

the problem of subpixel scanconversion is described by:

scanning of N' objects on a $M' \times M'$ (sub-)pixel raster display (for every pixel).

The meanings of the symbols are:

N the number of objects in the scene

N' the number of objects, which make a contribution to the pixel's color value

$M_1 \times M_2$ the number of pixels on the raster display

$M' \times M'$ the number of subpixels on the subpixel raster.

For a typical scene $N' \ll N \approx 5000$ objects and $M' \ll M_1 \sim M_2 \approx 1024$ pixels is valid. In general $N' < 10$ and $M' \leqslant 8$ is valid, too. This means a reduction of the problem by several orders of magnitude. We will come back to this in a later section of this paper.

Those objects, which contribute to the pixel's color value are determined in the object processor pipeline. This part of the architecture has as result a depth-sorted list of contributing objects for every pixel which is passed to the subpixel scanconversion.

The task of this unit cannot only be described by the above mentioned reduction but also in more general terms:

```
for every pixel
   for every object in the contribution list
      for every subpixel
      {
         compute the contribution of the object to the subpixel color value
      }
```

In this processing first the sequence of computations can be exchanged as well as second in every stage parallel or sequential processing is possible. The different possibilities of treatment lead to different architectures. Out of the variety of these possibilities we want to present two in detail.

However, the number of alternatives is reduced by the architecture given above:
A parallel processing of pixels is not appropriate because the pixels run through the object processor pipeline *sequentially*.
These points given will not result in a delay of the timing, if we take the chance to formulate a timing restriction for the subpixel scanconversion unit:

Each pixel should be treated by the subpixel scanconversion unit while the next pixel is processed by the object processor pipeline.
Unfortunately this timing constraint cannot be formulated exactly since the object processor pipeline works asynchronously.

In practice we add a further processing unit to the object processor pipeline. This results only in a marginal increase in computing time per pixel. A short example will clarify this: Suppose a number of N processors (equal to the number of objects), $M_1 \times M_2$ pixels on the raster display and an expected time t_P of processing per pixel. Then the resulting time to process the whole frame is: $D_1 = (M_1 \times M_2 + N) * t_P$. If we add one processor, as intended, this results in a time of $D_2 = (M_1 \times M_2 + N + 1) * t_P$. Typical cases include $N = 5000$ objects and $M_1 \times M_2 = 250.000$ pixel. This results in a marginal less difference (lower than one percent!).

This is the basis on which we want to present the two architectures, their drawbacks and advantages.

3. The Pipeline Architecture

As we pointed out, the task of subpixel scanconversion can be seen as a reduction of the task of scanconversion. In consequence we can transfer the solution of the problem, that is the architecture, too. This results in the use of a (simplified) object processor pipeline substituted for the black box subpixel scanconversion (figure 4).

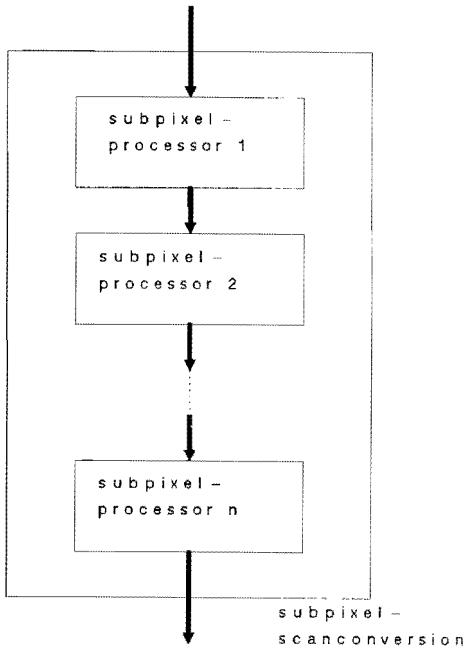

Figure 4: Subpixel Scan Pipeline

The timing will then be as in the object processor pipeline:

```
for every pixel
{
   1. load the object information from the sorted list into the processors

   2. pipelining of the subpixels through the object processors

      (a) (incremental) computation of the contribution of the object to the subpixel

      (b) shifting of the information to the next processor
}
```

As you can realize, the objects are treated sequentially unlike the subpixels, which are treated partially in parallel. The degree of parallelism depends on the number of subpixels. If we compare this solution to the "single processor" solution, which processes the whole problem sequentially, we will see that it gets worse with decreasing number of subpixels and increasing number of objects, respectively.

Suppose to be:

O the number of objects in the scene

t_L the load time for an object's information

t_B the processing time per subpixel and object.

A sequential solution will need:

$$t_1 = t_{seq} = O \times M' \times M' \times (t_L + t_B)$$

time to load and compute the color value for *one pixel.* On the other hand the proposed pipeline architecture will require:

$$t_2 = t_{pip} = O \times t_L + (O + M' \times M') \times t_B$$

Above all, there is the condition that a pixel should be subpixel-scanconverted before the next pixel leaves the object processor pipeline. Strasser stated the expected time between the leaving of two pixels to be $35ns$!

We can recognize that this time constraint can *not* be satisfied by todays technology. However, one typically needs $500ns$ per object for loading the data. With at most 10 objects per pixel, this would result in a load time of $5\mu s$.

If we alternatively look at the whole system as a system that consists of geometry processor, object processors and subpixel scanconversion, which is interconnected in such a way that it can act as a pipeline too, every stage has to perform its task in $20ms$. In this case the computation time per pixel results in

$$t_P = 80ns$$

if we suppose the same data as above. This time cannot be reached as well if we look at the considerations of this section.

4. The Parallel Architecture

The alternative concept to the discussed *one processor per object* is the use of *one processor per subpixel.* This means no more pipelining of the subpixels but parallel processing of the subpixels while sequentially processing the objects. In figure 5 you can see the associated architecture.

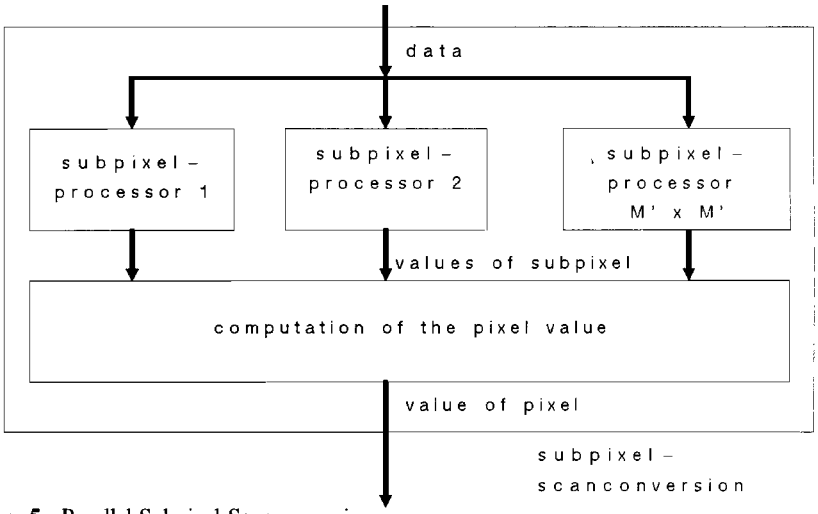

Figure 5: Parallel Subpixel Scanconversion

The timing of the processing will be:

```
for every pixel
{
   1. load the pixel data (which means: increment xy-register)

   2. for every object
      {
          (a) load the object's information

          (b) if the subpixel is not yet colored, compute the contribution of the object
      }
}
```

Since the pixel data as well as the object information is passed sequentially from the object processor pipeline this order of computation fits into the given architecture pretty well.

The main advantage is the consequence of the fact that one needs just a *fixed number* of processing elements, namely $M' \times M'$. In opposition to that the number of processors in the pipeline could not be fixed. As we already mentioned above, a fixation of the number results in a limitation of the complexity of the scene.

The resulting time needed for the loading and computing of one pixel results in:

$$t_3 = t_{par} = O \times (t_B + t_L)$$

A comparison of the times t_1, t_2 and t_3 with a fixed number of objects per pixel is shown in figure 6. One can see clearly the differences in quality of the different architectures.

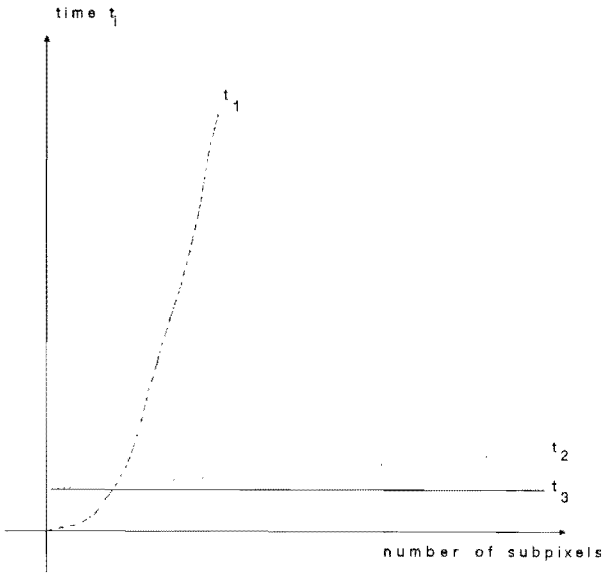

Figure 6: Times needed by the Different Architectures

But what are the time constraints for this approach?
If we again suppose the time per pixel to be $35ns$ and the maximum number to be 10 objects per pixel this results in $3.5ns$ for the computation of the subpixel values *and* the pixel value. Even in modern CMOS-technology this is the equivalent to 7 cycle times. This amount will surely be exceeded by our task.

If we follow the second philosophy of structuring the image processing as a pipeline, the timing condition leads again to $t_P = 80ns$. This constraint is more realistic as we will show soon. Hence we found an appropriate completion to the architecture given by Strasser, which performs the task of subpixel scanconversion.

From figure 5 follows that the pixel color value is a combination of the weighted subpixel color values. That means that in the unit "computation of the pixel value" a multiplication and addition of the red, green, and blue components is performed. Additional filtering can be added (included in the weigthing) without increasing the needed time.

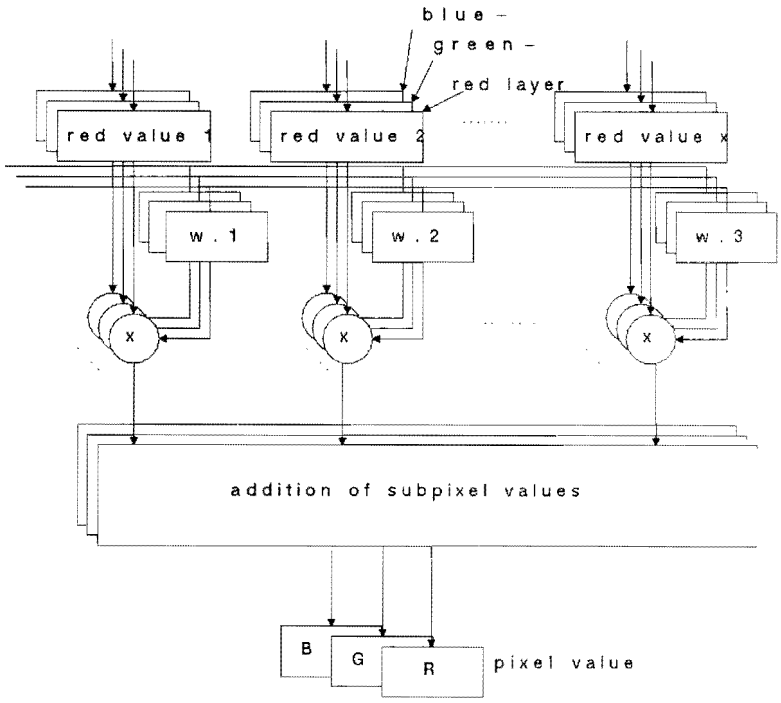

Figure 7: Computation of the Pixel Color Value

Details of this part of the architecture can be seen in figure 7. The "addition" unit can be implemented using one of the well known fast VLSI-algorithms. Loading the weights had to be done only once for a sequence of frames so it will not result in additional time needs.

A single processing element needs two registers: one for the "geometric" data of the actual pixel (x and y) and one, which determines the subpixel data. In addition we need an increment unit and some other registers to hold the geometry and the color of the object (figure 8).

The objects are treated in the order of the depth-sorted list, which is arranged from foreground to background (rising z-values). This leads to the advantage that the processing unit can stop calculating in the moment a subpixel color value is determined. Determining the contribution can be done by the well known operation of clipping (the edges of the polygon against the edges of the subpixel). By means of Bresenham's algorithm we can decide if an object makes a contribution even if it covers only a part of a subpixel. Some further results about the realisation, the necessary resolution M', and the quality of the results are in preparation.

Guided by the figures it is evident that the task and its subtasks are easily built. Therefore a VLSI-implementation is possible. We are of the opinion that such an implementation could hold the timing constraints. So the wish to generate images of highest quality in "real time" is not unrealistic.

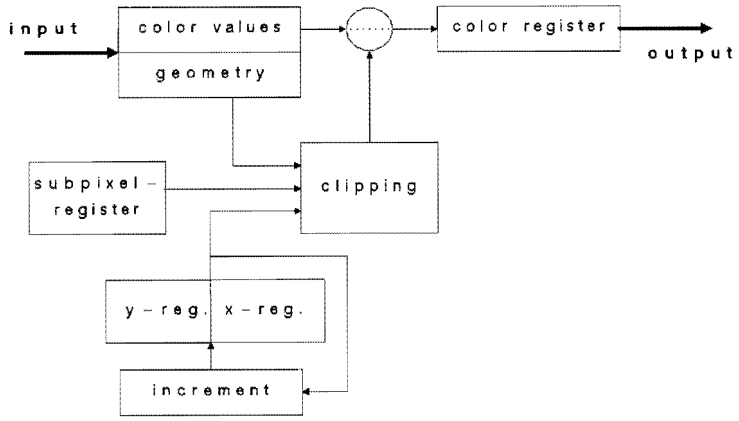

Figure 8: The Subpixel Processor

But we should not forget the mentioned drawbacks and that the functionality of shading is not yet implemented. The importance of this processing stage is evident for planar polygon based models. This and the extension to other primitives could be a starting point for further investigations.

5. Conclusions

For the task of subpixel scanconversion in an image generation system some solutions were developed. It could be shown that parallel processing fits better to the given architecture than pipeline processing. The presented architecture does not lead to a restriction in the used model or a limitation of the scene complexity. An integration into other systems is possible due to the pipeline character of the image generation system.

6. References

[CD-81] D. Cohen, S. Demetrescu:" A VLSI Approach to Computer Image Generation," *Technical Report*, Information Sciences Institute, University of Southern California (1981)

[Crow-77] F. Crow: "The Aliasing Problem in Computer-Generated Shaded Images," *Comm. of the ACM* **20** 11, November (1977)

[FiFR-83] E. Fiume, A. Fournier, L. Rudolph: "A Parallel Scan Conversion Algorithm with Anti-Aliasing for a General-Purpose Ultracomputer," *ACM Computer Graphics* **17** 3, pp. 141-150 (1983)

[Fuea-85] H. Fuchs et. al. : "Fast spheres, shadows, textures, transparencies, and image enhancements in Pixel-Planes," *Computer Graphics* **19** 3 pp. 111--120 (1985)

[FuPo-81] H. Fuchs, J. Poulton: "Pixel Planes: a VLSI-oriented design for a raster graphics engine," *VLSI-Design*, Third quarter (1981)

[Stra-85] W. Strasser: "A VLSI-oriented, highly parallel architecture for fast image generation of realistic scenes" in: K. Waldschmidt and B. Myhrhaug (eds.): *Microcomputers, usage and design, Proceedings of the EUROMICRO'85*, North-Holland (1985)

[Stra-86] W. Strasser: "A VLSI-oriented architecture for parallel processing image generation," in: Reijns, Barton (ed.): *Highly parallel computers* Elsevier Science Publications B.V. (North-Holland) (1987)

[Wein-81] R. Weinberg: "Parallel processing Image Synthesis and Anti-aliasing," *ACM Computer Graphics* **15** 3, pp. 55--62 (1981)