

A Hardware Accelerated Alternative for the Accumulation Buffer

Ares Lagae Frank Suykens Philip Dutré

Report CW 369, Januari 2003



Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

A Hardware Accelerated Alternative for the Accumulation Buffer

Ares Lagae Frank Suykens Philip Dutré

Report CW 369, Januari 2003

Department of Computer Science, K.U.Leuven

Abstract

In this paper we present a hardware accelerated alternative for the accumulation buffer. Multiple frames are accumulated in a hierarchical way, using texturing and programmable graphics hardware. The new algorithm has significant advantages over existing approaches and can be implemented on almost any recent consumer level graphics hardware architecture.

Keywords : accumulation buffer

CR Subject Classification : I.3.1: Computer Graphics - Hardware Architecture

A Hardware Accelerated Alternative for the Accumulation Buffer

Ares Lagae Frank Suykens Philip Dutré

K.U.Leuven University *

Abstract

In this paper we present a hardware accelerated alternative for the accumulation buffer. Multiple frames are accumulated in a hierarchical way, using texturing and programmable graphics hardware. The new algorithm has significant advantages over existing approaches and can be implemented on almost any recent consumer level graphics hardware architecture.

1 Introduction

Traditional raster based rendering systems are often extended with an accumulation buffer [1] which combines images rendered into the color buffer. The accumulation buffer can be used for effects such as full scene anti-aliasing, soft shadows from area light sources, motion blur, depth of field, etc. [6].

Current graphics hardware is able to generate images at rates of more than hundred frames per second, depending on scene complexity. Because of these high frame rates, the accumulation buffer is an excellent technique to improve on image quality while still maintaining interactive rates.

Today's consumer level graphics boards (e.g. the NVIDIA GeForce or ATI Radeon) do not have a hardware accelerated implementation of the accumulation buffer. Instead calculations related to the accumulation buffer are executed by the CPU. This involves the transfer of substantial amounts of image data from the graphics hardware to the CPU and back. Due to this overhead, a software accumulation buffer is not suited for interactive applications.

In this paper, we present a simple but effective alternative for the accumulation buffer. By accumulating images in a hardware accelerated hierarchical way, the technique speeds up computations involved with the accumulation buffer and enables image accumulation in interactive applications.

After discussing some alternate approaches to accumulate images in section 2, we introduce the concept of accumulation trees in section 3. Implementation details and results are given in section 4, before concluding in section 5.

*{ares.lagae,frank.suykens,philip.dutre}@cs.kuleuven.ac.be

2 Previous approaches

The only consumer level hardware implementation of an accumulation buffer we know of, is 3DFX's TBuffer. Unfortunately, 3DFX has ceased all activities and hardware with a TBuffer is not available anymore.

Images can be accumulated with hardware acceleration using blending [6]. All frames contributing to the accumulated image are rendered sequentially. For each frame, the alpha value of each vertex is set to the contribution of that frame in the final image. During rasterization, each fragment will receive this alpha value. The color buffer is not cleared while rendering the sequence of frames. Fragments of different frames are accumulated in the color buffer by the hardware. To ensure that each pixel in the final image is the accumulated result of only the corresponding visible fragments in the different frames, visibility information has to be calculated before color writes to the color buffer take place when rendering a single frame.

The major disadvantage of this technique is the limited precision of the color buffer. Color components are usually represented using only 8 bits. When multiplying color components with (small) alpha values, a lot of precision is lost. Because of this, accumulating images with blending produces inaccurate results even when combining a small number of frames. It becomes unusable when combining 16 or more images.

3 Accumulation Trees

3.1 Accumulation Nodes

The basic building block of our technique, an *accumulation node*, combines a small number of frames (e.g. 2 or 4) with equal weights. To accumulate two frames, the first frame is rendered into the color buffer. The content of the color buffer is then transferred to a texture. Because the color buffer and the texture are both located in the memory of the graphics hardware, this is a fast operation. In a similar way, the second frame is stored in a texture. Then a rectangular quad parallel to the viewing plane is drawn using an orthogonal projection. The quad is textured with both textures using multi-texturing. For each rasterized fragment of the quad, a fragment program averages color samples of both textures. By choosing the texture dimensions equal to those of the viewport, a one-to-one mapping between texels, rasterized quad fragments, and screen pixels is achieved.

This algorithm has several advantages over a software accumulation buffer and accumulation with blending:

- Storing intermediate results in texture memory eliminates the need to transfer image data from the graphics hardware to the CPU and back.
- Performing accumulation operations on 2D image data rather than on 3D geometry avoids visibility problems.

- Calculations in fragment programs are typically performed in high precision, eliminating possible accuracy problems.

Given a graphics hardware architecture, it is often possible to implement several types of accumulation nodes, each with its own number of inputs (images), for example two, three, or four. However, extending this algorithm to combine an arbitrary large number of frames is impossible: the graphics hardware has a limited number of texturing units and limited texture memory, and fragment programs have a limited number of instructions.

3.2 Accumulation Trees

Although extending the algorithm to an arbitrary number of inputs is not possible, accumulation nodes can be used as basic building blocks for a more general algorithm to average out a large number of frames.

Results of accumulation nodes can also be stored in textures. These textures can then be used as input for another node. This way, a tree is constructed in which the leaf nodes contain individual frames, the root node contains the accumulated image and the other nodes contain temporary image data. Non-leaf nodes correspond to accumulation nodes.

The branching factor of the tree is determined by the number of inputs of the accumulation nodes. For example, when using only accumulation nodes with two inputs, the *accumulation tree* is a binary tree (figure 1a). In order to produce an accumulated image in which each frame has an equal contribution, the tree has to be complete (all leaf nodes have to be at the same depth and the branching factor has to remain constant). Figure 1b shows an example of an incomplete tree. Nodes with different branching factors (number of inputs) can be mixed, but to produce a correct average, the branching factors of all accumulation nodes in a single level of the tree have to be identical and, again, the tree has to be complete (figure 1c). In figure 1d an example of an accumulation tree with different branching factors per level is depicted.

The requirements of completeness and equal branching factors per level place a limitation on the number of frames that can be combined. A number of frames N can be accumulated if N can be factored in $\prod_{i=1}^n b_i^{l_i}$, where n is the number of different node branching factors allowed in the tree, b_i is the i th branching factor and l_i gives the number of levels in the tree with nodes using branching factor b_i . For figure 1a and 1c these factorizations are 2^2 and $2^1 4^1$ respectively. In section 3.5 we will extend the method to allow for the frames to have different weights and, as such, relax the limitation on the number of frames.

To render the accumulation tree, it is traversed in postfix order. For each leaf node, a frame is rendered and stored in a texture. Accumulation nodes take the textures produced by their children as input and produce a single image as described before. This image is stored in a texture except when the node is the root of the tree, then the image can remain in the color buffer and be displayed. The complete algorithm is given in figure 2.

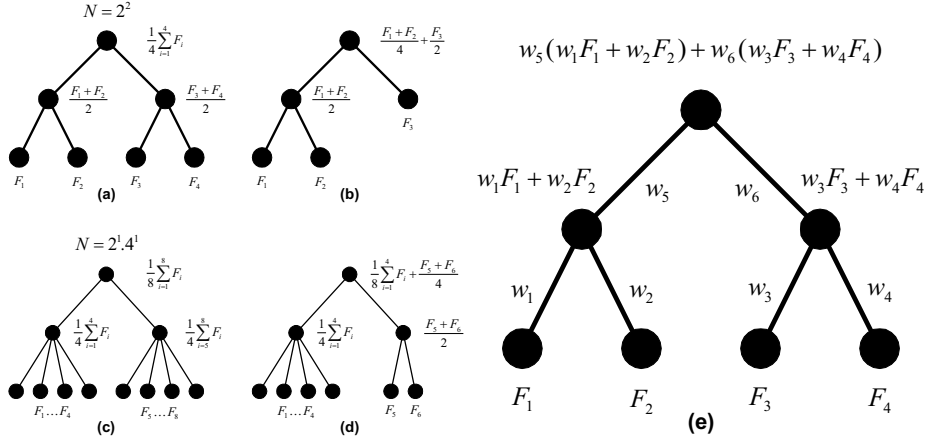


Figure 1: Various accumulation trees. Leaf nodes are individual frames and the root node is the accumulated image.

3.3 Complexity

An accumulation tree with N leaf nodes will never have more than $2N$ nodes and each node requires exactly one render pass. Consequently, the complexity of this method is linear in the number of input frames, just as a software accumulation buffer and accumulation using blending.

After an accumulation node has been evaluated, the textures used by its children can be reused. For an accumulation tree with a constant branching factor b and N leaf nodes, the number of required textures is given by $(b - 1) \log_b N + 1$, where $\log_b N$ is the depth of the tree. This number is equal to the maximum size of a stack needed to evaluate a tree in postfix order (see figure 2).

3.4 Accuracy

The accuracy of this method depends on the internal precision of fragment programs, the precision of the textures used to store intermediate results and the tree branching factor.

For an accumulation node with two inputs, a fragment program architecture with 9 bits precision (recent graphics boards, like the GeForce FX or the Radeon 9700 do better) and a texture format using 8 bits per color component, the maximal absolute error in the result is 0.5 (using a $0 \dots 255$ range). When the least significant bit of the input values differ, the binary fractional part .1 resulting from the division by two is lost. For an accumulation tree composed of three binary accumulation nodes (figure 1a), the absolute error is $(0.5 + 0.5)/2 + 0.5$. Each level adds an additional 0.5 to the absolute error. For a binary accumulation tree with N leaf nodes, the maximum absolute error is $0.5 \log_2 N$.

Under the same conditions, a node with four inputs having the same error bound can be constructed. Fragment program architectures with higher precision allow

```

// the next free texture
// used as a pointer in a texture stack
nextTexture = 1

// the next frame to render
nextFrame = 1

// render the accumulation tree
procedure renderAccTree(node)
  if isLeaf(node)
    renderFrame(nextFrame++)
    copy color buffer to nextTexture
    nextTexture++
  else
    // an accumulation node
    renderAccTree(leftChild(node))
    renderAccTree(rightChild(node))
    // top of stack contains both child textures
    combineTextures(nextTexture-2, nextTexture-1)
    if not isRoot(node)
      copy color buffer to nextTexture-2
      nextTexture--
    end if
  end if
end procedure

// combine two textures to the color buffer
procedure combineTextures(texture1, texture2)
  set orthogonal projection
  clear color buffer
  enable texture1 and texture2
  enable fragment program
  draw textured quad
  disable fragment program
end procedure

```

Figure 2: The accumulation tree algorithm.

the construction of nodes with a lower error bound and more inputs. Also, storing intermediate results in higher precision texture formats will have a positive impact on the error. Therefore, the maximum error, $0.5 \log_2 N$, also provides a bound for these cases.

The quality of images accumulated with this technique is superior to that of accumulation using blending (figure 3). For example, combining 256 images using blending, which has a linear error bound, results in a black image. This is because blending calculations are generally performed on 8 bit values, and a multiplication by the alpha value $1/256$ loses all significant bits when stored in an 8 bit color buffer. In contrast, accumulating the same number of images with an accumulation tree results in a worst case error of $4/255$. Of course a regular (software) accumulation buffer using for example 32 bit per color component will be more accurate.

3.5 Adding Weights

A *weighted accumulation tree* is an accumulation tree in which each edge has an associated weight. Fragment programs of accumulation nodes are altered

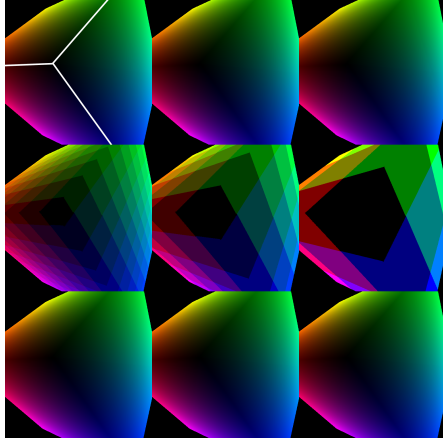


Figure 3: Accumulation of (from left to right) 32, 64 and 128 identical RGB color cube images using a regular accumulation buffer (top row), blending (middle row) and an accumulation tree (bottom row). Note the artefacts in the middle row.

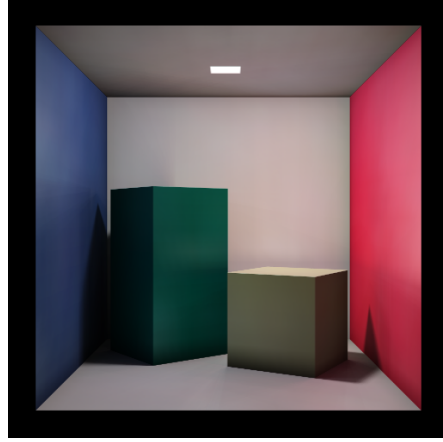


Figure 4: A Cornell box with hardware accelerated global illumination. To acquire this result, 1024 images were combined using an accumulation tree. Note the soft shadows and indirect lighting (e.g., on the ceiling).

in order to calculate the weighted sum of their inputs rather than assuming uniform weights. The total contribution of an input frame to the final image is the product of the edge weights along the path from the root node to the corresponding leaf node (figure 1e). Suitable edge weights should be chosen in order to produce the desired combination of input frames.

To fully utilize the dynamic range of the color components in the textures and to avoid possible accuracy problems, the weights of all input edges of an accumulation node should sum to one.

The accuracy of a weighted accumulation tree depends largely on the accuracy of the fragment program architecture. Similar to accumulation using blending, multiplications with small weights might cause the loss of many significant bits if the precision of the fragment programs is not sufficient.

Furthermore, by also allowing zero valued weights, we can easily relax the limitation on the number of frames that can be combined. To accumulate an arbitrary number of frames N , an accumulation tree having l leaf nodes, with $l \geq N$ is constructed. This tree will have $l - N$ leaf nodes with zero weight. If the leaf nodes are arranged in such a way that leaf nodes with zero weight are grouped together on one side of the tree, an entire branch of the tree can be cut away. Note that allowing regular accumulation trees to be incomplete (figure 1b) or to have different branching factors per level (figure 1d) can also simulate the effect of different weights.

4 Results

We implemented the accumulation tree algorithm on an AMD 1500 MHz computer using a GeForce 3 graphics card, OpenGL [5] and the OpenGL extensions [2] `ARB_multitexture`, `NV_texture_rectangle` and `NV_register_combiners`. To achieve higher accuracy, `ARB_fragment_program` and/or `ATI_texture_float` could be used. Our implementation supports accumulation nodes with a branching factor of two and four (same accuracy, but faster). Note that it is not necessary to use recursion or to explicitly model the tree structure. Although it can be calculated as a preprocess, the postfix tree traversal can also be simulated using a stack for example.

The popular OpenGL examples from ‘The Red Book’ [6] using an accumulation buffer were re-implemented with accumulation trees. For example, our implementation of the *scene anti-aliasing* example achieves a frame rate of about 50 frames per second at a resolution of 512×512 , while the standard implementation using a software accumulation buffer runs at about 0.59 frames per second, at the same resolution. In both cases, 8 frames were accumulated. Other examples showed a comparable speedup.

To show that accumulation trees are also suited to combine a large number of frames, an extension of the instant radiosity technique presented in [3] was implemented using accumulation trees. An approximation for the global illumination in a scene is calculated by placing a large number of virtual light sources in the scene and accumulating the direct illumination for each light source. More details about this method can be found in [4]. An example using 1024 images is shown in figure 4. This technique could not have been implemented using accumulating with blending.

Table 1 compares the performance of a software accumulation buffer, accumulation using blending and accumulation trees, for different input sizes. The performance was measured in frames per second for a test scene, at a resolution of 512×512 . Accumulation trees are roughly 100 times faster than a software accumulation buffer and about two to three times slower than accumulation using blending, but recall that in practice accumulation using blending is restricted to a few frames. Accumulation trees using nodes with higher branching factors are faster because they require less rendering passes (as they have fewer nodes).

5 Conclusion

In this paper we have introduced accumulation trees, a novel method to accumulate images. The technique is fully hardware accelerated and is therefore significantly faster than a software accumulation buffer. In contrast to accumulation with blending, accumulation trees can also handle large input sizes and have a higher accuracy. This is due to their logarithmic memory requirements and logarithmic error bound. Accumulation trees enable the use of the accu-

N	Acc. Buffer	Blending	Acc. tree (2)	Acc. tree (4)
2	1.77	344.56	254.71	-
4	1.06	194.28	105.51	132.67
8	0.59	103.72	48.59	-
16	0.31	(53.72)	23.37	30.22
32	0.16	(27.34)	11.47	-
64	0.082	(13.79)	5.68	7.39
128	0.041	(6.92)	2.83	-
256	0.021	(3.47)	1.41	1.84
512	0.010	(2.00)	0.70	-
1024	0.0052	(1.00)	0.35	0.46

Table 1: Performance of a software accumulation buffer, accumulation using blending and accumulation trees with branching factors 2 and 4, in frames per second. Values between brackets indicate poor image quality.

mulation buffer in interactive applications and are relatively easy to implement on commodity graphics hardware.

Each generation of graphics hardware tends to support more texture units and new fragment program architectures that have a higher internal precision. These developments will allow to construct accumulation nodes with even lower error bounds and higher branching factors. This will benefit both the speed and the accuracy of future implementations of the accumulation tree algorithm.

References

- [1] Paul Haeberli and Kurt Akeley. The Accumulation Buffer: Hardware Support for High-Quality Rendering. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 309–318. ACM Press, 1990.
- [2] Silicon Graphics Inc. OpenGL Extension Registry (specification documents, available from <http://www.sgi.com>).
- [3] Alexander Keller. Instant radiosity. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 49–56. ACM Press/Addison-Wesley Publishing Co., 1997.
- [4] Ares Lagae. Interactive Realistic Rendering using a Hardware Programmable Graphics Pipeline. (in dutch). Master’s thesis, Katholieke Universiteit Leuven, Leuven, Belgium, 2002.
- [5] Mark Segal and Kurt Akeley. The OpenGL (R) Graphics System: A Specification (Version 1.3). 2001.
- [6] Mason Woo, Jackie Neider, Tom Davis, and Dave Shreiner. *OpenGL Programming Guide: The Official Guide to Learning OpenGL. Release 3*. Addison-Wesley, 1999.