Fast Rendered Animation Compression using Point of View Data

David Coulthurst

Department of Computer Science
University of Bristol
England

1 Abstract

As the demand for more realistic rendering of images increases, and the hardware necessary to achieve this becomes more expensive, rendering has become an off site and on demand issue. A major challenge of this remote rendering is the rapid lossless transmission of the results back to the client. Remote rendered animations contain Point of View information not traditionally used for animation compression. This paper presents two novel algorithms for fast animation compression based on frame estimation from point of view movement across frames. A gain of 40-50% lossless compression is achieved.

Keywords: Image Compression, Remote Rendering, Rendering on Demand, Motion Compensation

2 Introduction

Rendering in high detail is an extremely computationally intensive operation. Despite the advent of modern GPU's, producing individual frames of an animation of this quality can take hours or even days on a desktop PC. Parallel processing can be used to significantly reduce overall rendering times, but getting a cluster of computers to work together is a complicated process [CDR02], and the costs of even a moderately sized cluster are typically beyond the financial resources of most small media companies. This is especially true as such expensive clusters are not needed continuously, and thus stand idle for significant periods of time. This has led to the growth of "render farms" with dedicated companies providing large rendering services to clients when they are needed.

In addition to parallel rendering, exploitation of the human visual system has also been shown to significantly reduce overall rendering time, without a perceptual loss in perceived image quality [YPG01,CCW03,SDL*05]. The "Rendering on Demand (RoD)" at the University of Bristol combines work on visual perception and techniques for parallel rendering, with an aim of creating a real-time remote rendering system for high fidelity imagery [CC02].

A key issue that still needs to be addressed, however, is how to deliver the remotely rendered frames of an animation to a client rapidly, and without any loss of detail. There are, of course, a number of compression algorithms available. The most common of these are the well known single image JPEG standard [W91], and video compression standard MPEG [G91, Mpeg94, Mpeg98]. Both JPEG and MPEG operate on discrete cosine transformed (DCT) pixel blocks, 8 by 8 for jpeg, 16 by 16 for MPEG. These are then rounded, and this combined with the DCT leaves a much smaller piece of data to record. However, the rounding loses data, resulting in lossy compression. So although a high rate of compression is achieved, the compressed image contains less data than the original. Figure 1 (a) shows the original high-fidelity rendered image of a fire extinguisher, while (b) shows the same image compressed under JPEG, with a noticeable loss in data and thus quality. The effect is the same in MPEG.

Figure 1: Original uncompressed image

Figure 2: JPEG compressed image

MPEG uses a mix of Inter-frame and Intra-frame compression by matching these blocks. Three types of frames are used – Intrapictures (I), Predicted pictures (P), and Bidirectionally Interpolated pictures (B). The I frames provide random access points, but have less compression than the other frames. P frames are formed from references to previous frames (I or P frames) and thus are also used as references for future predicted frames. B frames are created from both past and future I and P frames, but are themselves not used as references. A typical encoding could have frames:

I B B P B B P B B P B B I

When a frame is created from references, it is with the use of motion compensation. In the case of MPEG this

means that 16 by 16 blocks are searched for in previous I and previous and future P frames. It assumes that the current frame can locally be modelled as a translation of the picture at some previous time. The term locally is used because the blocks can be from different positions in different previous frames. These block matches are found using a brute force search - it is a brute force Block Matching Algorithm (BMA).

A lossless motion compensated compression algorithm is presented by Guenter et al. [GYM93] that uses the motion of each object in the scene. They calculate and store all the information needed to compute the optical flow vector for each pixel. The decoder can reconstruct the frame by backprojecting each picture from the previous frame. Although this allows for a wide range of motions including translations, scalings and rotations, the overhead associated is large compared to image-based motion estimation schemes.

Agrawala et al. [ABC95] present an approach that uses object movement in a scene similar to [GYM93], but combined with BMA techniques. The algorithm first calculates the optical flow field based on object movement. This is then combined for blocks of pixels, to create a projective matrix that best encodes the motion of the pixels in the block, determined using a least squares algorithm.

Jpeg-ls [WSS98] is a newer standard than JPEG, which allows lossless and near lossless compression to be achieved. It is as low complexity version of the universal context modelling paradigm. It matches the modelling unit it uses – which models how one pixel relates to the previous pixel in the image – to a simple coding unit – which codes the model into a file format.

In this paper we discuss how Point of View (PoV) data, which is available, associated with the camera view, in computer animations, may also be used to reduce the number of pixels that need to be transmited between a remote render farm and its client. PoV data is used to work out how the viewpoint has moved between frames. This in turn allows estimates of each frame to be created from the previous frame. By comparing these estimates to the real frame, compression can be achieved by only sending the data not contained in an estimate. It can be considered as a non-brute force BMA, unlike MPEG. However instead of using the motion of objects in the scene, as in [GYM93, ABC95], it only uses the PoV movement derived from the camera. This results in an algorithm that is simpler and less computationally intensive.

3 Scene Estimation Algorithm

The movement of the PoV can be considered as a mix of translation and rotation about each axis. The x-axis is considered to be horizontal, y-axis to be vertical, and z-axis to perpendicular to the view plane (Figure 3), with the point in the centre of the current frame to be (0,0,0).

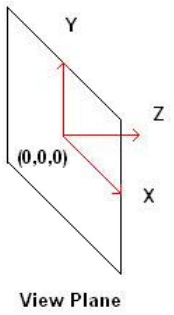

Figure 3: Scene approximation co-ordinate system

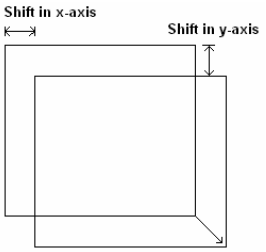

Figure 4: Point of View movement along x-axis and y-axis

Movement of the PoV along the x-axis m units corresponds to the appearance of the contents of the scene moving in a negative direction on the x-axis, by a factor of m. For example, by moving the PoV left, the scene will appear to move right(Figure 4). A frame translated the correct amount along the x-axis thus gives an estimate of the frame to use for compression. Translation of the PoV in the y-axis can similarly be estimated with opposite translation of the frame in the y-axis.

Movement in the z-axis corresponds to moving in and out of the scene (Figure 5). In the case of moving into the scene (a positive translation in the z-axis), the scene appears to have grown larger, scaling out from the centre of the frame.

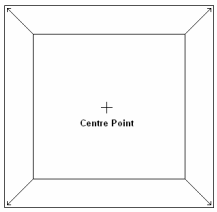

Figure 5: Point of View movement along z-axis

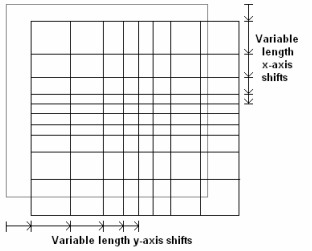

Figure 6: Rotation using non-uniform adjustment

Rotations around the x-axis and y-axis are handled by translations of the frame. Rotations require that the edges of the frame are translated more than the centre. For example, a rotation around the y axis will have the right and left edges of the frame translate more than the centre translates along the x-axis (Figure 6). Rotation about the z-axis is not approximated in the system, as no fast algorithm was devised.

For the highest level of accuracy, ideally all of these operations would be done using floating point calculations to generate the new pixels of the estimate. However, for a fast algorithm, integer calculations are more appropriate. The image is divided into a grid, and the amount of pixels each square should be translated by is calculated. For example Figure 7 shows a small scaling, using a five by five grid. The translation and rotation operations detailed above can all be implemented using a grid.

+2,+2	+1,+2	+0,+2	-1,+2	-2,+2
+2,+1	+1,+1	+0,+1	-1,+1	-2,+1
+2,+0	+1,+0	+0,+0	-1,+0	-2,+0
+2,-1	+1,-1	+0,-1	-1,-1	-2,-1
+2,-2	+1,-2	+0,-2	-1,-2	-2,-2

Scaling using 5 squares to a side

Figure 7: Small scaling using a five by five grid

To get as accurate an approximation, while retaining the use of integer operation when actually generating the estimate, the following algorithm was used. Each 'square' refers to an entry in the matrix E, but is referred to as a square as it is the operation corresponding to an 8 by 8 square of pixels.

- Calculate Scene distance variable c (based upon how far away visible objects are and how many squares used for estimation matrix, can be combined with values for movement along and rotation around axes before sending to server).
- Set up overall estimation float matrix E, all values set to (0,0)
- Calculate scaling float matrix S - each square is (X,Y), where:
X = No. of squares from centre square in x co-ordinates * distance moved along z-axis * c).
Y = (No. of squares from centre square in y co-ordinates * distance moved along z-axis * c).
- Sum E and S into E.
- Calculate movement float matrix M - each square is (X,Y), where X = movement along x-axis * c and Y = movement along y-axis * c.
- Sum E and M into E.
- Calculate rotation matrix R. For each axis, if the image is to be rotated such that it brings an edge forward, start from the opposite edge. For example positive rotation around the y axis brings the right edge of the frame forward, so calculate X co-ordinate shifts of pixels from the left is given by:
X = (1 + q2) * c / 10
Where q is the number of squares in from the edge.
- Sum E and R into E.
- Cast E from a floating point matrix to an integer matrix I.
- Apply I to previous frame to get and estimate frame (as in Figure 8).

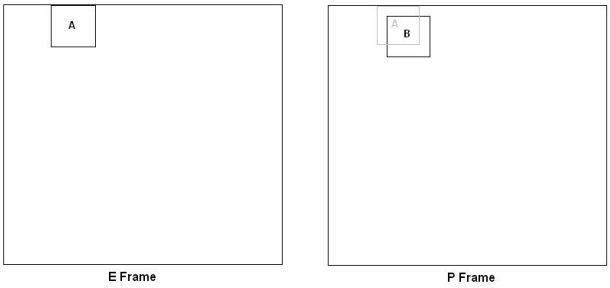

Figure 8: Sub-region re-sampling to generate estimates

Figure 8 shows how this is used to generate an estimate. For each square in the estimate frame the pixel values are generated from a square in the previous frame. So in the estimate frame the square A, is generated by copying the pixels of the square in the previous frame, offset by the amount in the integer matrix – square B. If more than one estimate is required the procedure is repeated.

4 Compression Algorithms

Two compression algorithms were developed. Both work on each band of an image at a time, with one byte samples per pixel per band. Both work by comparing the data frame - the frame to be compressed - with one or more estimate frames – created from the previous data frame using the scene estimation algorithm. The user can create the same estimate frames as the estimates the server has created. The user just runs the same code that decides which estimates are created that the server uses. As the PoV motion data is the same by definition the same estimate frame are created.

Position	-	Data Frame	Estimate 1	Estimate 2
0		43	23	23
1		84	185	185
2		145	186	186
3		35	7	7
4		23	64	64
5		98	98	47
6		185	86	86
7		146	146	25
8		54	54	54
9		92	186	92
10		51	51	51
11		214	214	214
12		153	153	153
13		120	120	120
14		195	45	45
15		35	65	37

Table 1: Example data and estimates

The first algorithm, the gap-match algorithm works by considering the data frame compressed and the one to four estimate frames as a stream of integers. The data is scanned through for sufficiently long matches between the data frame and one of the estimates. A match is sufficiently long, if encoding the match requires less bytes than recording the data.

The encoding works by recording the gap to the next match, and then the number of bytes of a match. One byte is used for each of the values. If using more than one estimate, the estimate the match comes from must also be recorded. The result of this is that a match must be four bytes long before any compression is achieved. Gaps or matches that are longer than 255 are handled by having a gap-match pair of 255,0 or 0,255 respectively.

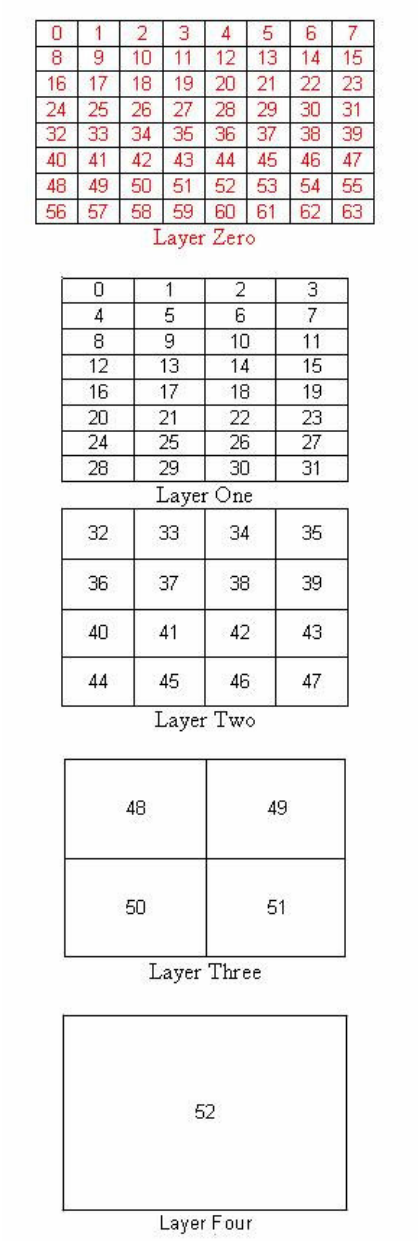

Figure 9: Tree structure showing how each number maps to a region of samples

Table 1 shows a small example of a portion of the stream of data. The matches at positions 5 and 7 for estimate 1 are ignored because they are too short to give compression. In this example the match from 8 to 13 in estimate 2 would be chosen. The gap-match pair of 8,6 would be recorded, then the data from positions 0 to 7 recorded. Decompression is simple. The gap-match pair is read. The number of bytes given by the gap is read in. The number of bytes given by the match is read from the estimate created on the user.

The second algorithm uses a tree-based implementation and uses one byte to describe a match. This means that the minimum size of a match is two, rather than the four needed for the gap-match algorithm. The image is split into eight by eight pixel squares. To represent possible combinations of matched squares, the tree structure in Figure 9 is used. Layer Zero represents the actual 64 samples in the eight by eight square. However, we are using matches or length two or more. Layer Two through Four show the numbers used and the regions they correspond to. This tree structure uses six bits to record the number of a matched region. This means that there are two bit left from a byte to use to denote which estimate frame the region is from. Two bits denotes four possible frames, one of which must be no-match rather than an estimate frame.

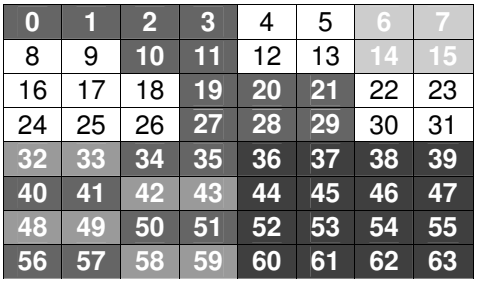

Figure 10: Example 8 by 8 square

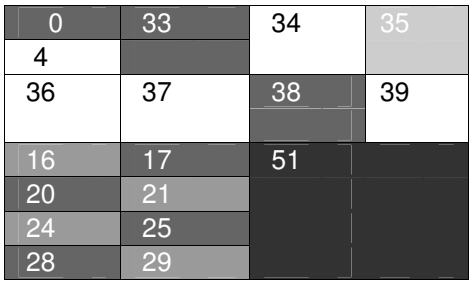

Figure 11 Encoded Regions

A byte can be used at the beginning of the encoding of the square to denote how many regions we are specifying. Then we can not specify some regions, and have this denote another estimate frame. This gives four estimates frames and no-match. This idea can be taken even further. The frame that has the highest number of regions (this can be an estimate frame or no-match) is denoted by not specifying a region. This means the

largest quantity of regions for a square are written by implication rather than use of actual bytes. To do this there must be an encoding to show which estimate frame or no-match is to be denoted by not specifying. This can be combined with the number of regions, and encoded into a single byte. No-match is denoted 0, estimate frames from 1 to 4. The number of the frame that is inferred is removed, and higher numbers shifted down. So if frame 2 was to be ignored, frame 3 would be denote 2, and frame 4 denoted 3. This is so the number stay in the range 0 to 3.

Figure 10 shows a typical 8 by 8 square, with the white regions denoting no-match, and the various grey denoting matches to different estimates. Figure 11 shows the regions that are actually encoded, and the numbers denoting those regions. So the square could be encoded thus. First the inferred frame (in this example frame 3 as it is the most frequent), and then the number regions specified are recorded. After this the frame number–region pairs are recorded. {x,y} denotes 2 numbers encoded as a single byte. The encoding would be:

{3,11} {0,4} {0,34} {0,36} {0,37} {0,39} {1,35} {2,16}{2,21} {2,24} {2,29} {3, 51}

This would be followed by the 18 bytes of unmatched data. So 64 bytes is encoded into 30 bytes (12 bytes of encoding data and 18 bytes of non-matched data).

5 Results

The system was tested on two animations, Animation 1 and Animation 2. Both of the animations are the work of Veronica Sundstedt, as part of a paper on Visual Attention [SDL*05]. They were chosen as they are both very high detail rendered images, typical of the type of animation the RoD system is aimed at.

Animation 1 (frame 1 is shown in Figure 12) consists of moving from a room into a corridor, and then turning around a corner. Animation 2 (frame 1 is shown in Figure 13) consists of moving down a corridor at a faster pace and then moving through a door out into a room. Both are 600 by 600 pixel tiffs, with three bands (Red, Green Blue), each band having 1 byte per sample.

Tables 2 and 3 show the time per frame and the percentage compression achieved for each compression method and number of estimate frames. Figure 14 Is a graphical representation of the overall system results data.

Figure12: Animation1 Frame 1 [S05]

Figure 13: Animation 2 Frame 2 [S05]

	Animation 1			
	Tree Based Method		Gap-Match Method	
	Time per frame	Compression	Time per frame	Compression
	(ms)	(percentage)	(ms)	(percentage)
1 Estimate	376	28.0569	194	30.26003
2 Estimate	391	29.523575	206	30.406431
3 Estimate	458	37.79552	239	36.5382
4 Estimate	537	42.440067	256	39.257

Table 2: Overall system Results Animation 1

	Animation 2			
	Tree Based Method		Gap-Match Method	
	Time per frame	Compression	Time per frame	Compression
	(ms)	(percentage)	(ms)	(percentage)
1 Estimate	308	24.23625	197	26.01551
2 Estimate	391	26.225758	211	26.412603
3 Estimate	463	33.940395	242	31.802519
4 Estimate	537	37.714745	262	33.65017

Table 3: Overall System results Animation 2

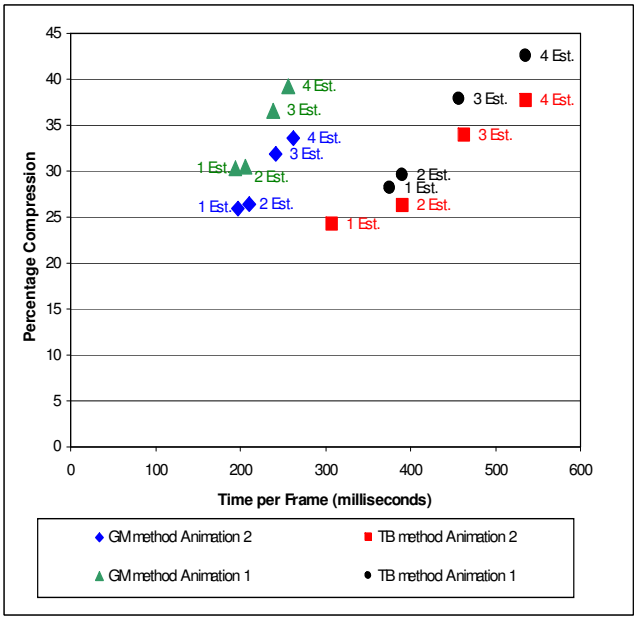

Figure 14: Overall system results presented graphically

The results in Table 2 and Figure 12 show us that the compression rates can be achieved in between 0.2 and0.5 of a second per frame. This is short of what is needed for real-time, by about a factor of ten. However, there are two important points to note. Firstly, the timing results were gathered from a high-end desktop. The full RoD system however is envisaged as running on a server, with far more processing power than a desktop. Secondly the algorithm is implemented in a high level language (Java). It is expected that a low level language version, or even a hardware version would be used if implemented commercially.

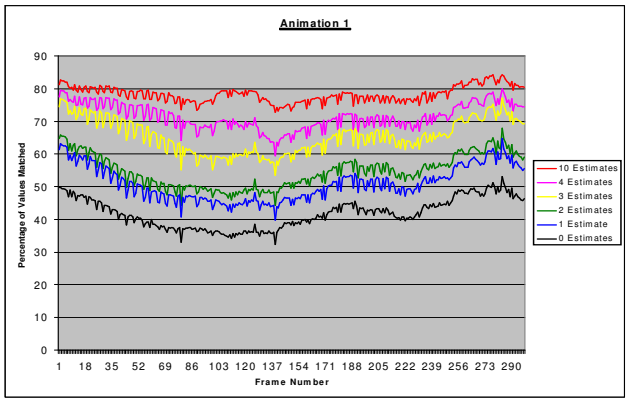

Figure 15: Matching results for Animation 1

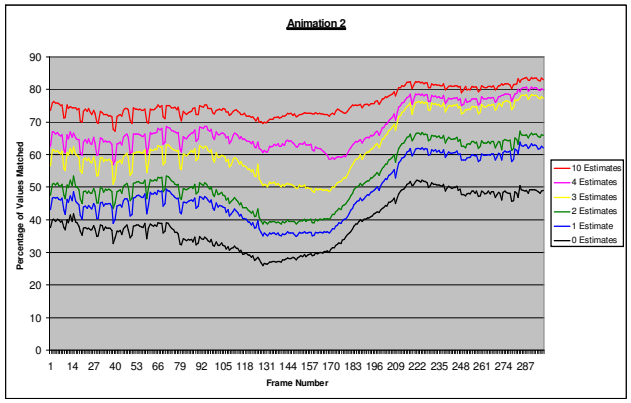

Figure 16: Matching results for Animation 2

Figures 15 and 16 show the percentage of matches for each animation, depending on the number of estimate frames used. 10 Estimates is included because above this, there is insignificant increase in matching. Figure 17 shows pictorially a typical level of matching for a frame. The black areas are where all three channels (R,G,B) match. The pink yellow or turquoise areas have 2 bands matching. The red, green and blue areas have 1 match. The white areas have no matches in and bands.

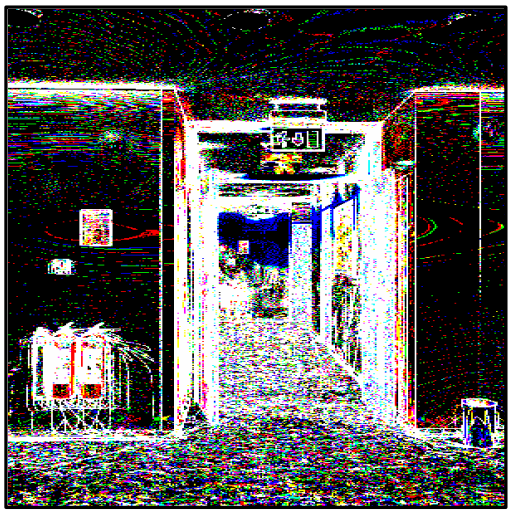

Figure 17 – Graphical Representation of a typical match

These results show several things. Firstly, there is a moderate amount of matching between each previous frame and the next, ~40% in Animation 1, and ~35% in Animation 2. Secondly, the use of the scene approximation algorithm increases the amount of matches, and the more estimates we use, the more matches are achieved. With four estimate frames (the maximum used in the compression algorithms), around 70% matching for Animation 1, and around 65% matching for Animation 2 was achieved. Thirdly the increase in matches with increased number of estimate frames is not linear, and slows off (the increase in matches between 3 to 4 estimations, and 4 to 10 estimations is roughly the same).

6 Conclusion

Designing a compression algorithm for a RoD system has the opportunity to take advantage of scene data not present in video compression. The results show compression based on PoV movement can field significant benefits.

The first new algorithm designed is the scene approximation algorithm. The algorithm performs well matching highly (Table 2 and 3, Figure 14), although as a stand alone device it is not that helpful. The output s frames are not smoothed, and if considered as stand alone images appear split with lines where the edges of the moved samples are. However this is not what it is intended for, it is designed to be the input stage to the compression algorithms, and as this it performs very well.

The compression algorithms both show potential for expansion. The compression achieved is achieved without the use of conventional image compression techniques such as that in JPEG-LS [WSS98] This is significant as it should allow the techniques used here to be combined with current work on image compression to gain a level of compression higher than either separately. Also useful is the fact that the time spent carrying out compression can be traded off against the amount of compression achieved. The performance of the algorithms is significant, achieving 40-50% compression (Table 2 and 3, Figure 14), which is comparable to the rates achieved in[WSS98]. Coupled with the scene estimation algorithm they provide a significantly increased amount of compression over a traditional video conferencing (running losslessly) style of streaming.

To achieve real-time speeds, it will be necessary to run the compression algorithms on a server rather than a desktop computer, or implement them in hardware, however this is the expected environment of the system. One flaw in the scene estimation algorithm is the inability to handle rotations in the z-axis. This is an issue that does need addressing, although a fast integer approximation of the rotation operation could not be devised in the time.

7 References

[ABC95] M. Agrwala, A.C. Beers, N. Chadda. "Model-based motion estimation for syntetic animations." In: *Proceedings of the 3rd ACM International Conference on Multimedia '95.* ACM, New York, pp. 477-488.

[CC02] A. Chalmers and K. Cater. "Realistic Rendering in Real-Time". In: *Euro-Par 2002 Parallel Processing,* pp 21 – 28. Springer Lecture Notes in Computer Science, August 2002

[CCW03] K. Cater, A Chalmers and G. Ward. "Detail to Attention: Exploiting Visual Tasks For Selective Rendering", In: EuroGraphics Symposium on Rendering 2003, pages 270—280. ACM, June 2003.

[CDR02] A.Chalmers, T. Davis and E. Reinhard. "Practical Parallel Rendering", AK Peters Ltd. July 2002.

[W91] G.K. Wallace, *"The JPEG still picture compression standard"*, Commun. ACM 34 (4) (April 1991) 30--44.

[WSS98] arcelo J. Weinberger, Gadiel Seroussi, Guillermo Sapiro. *"The LOCO-I Lossless Image Compression Algorithm: Principles and Standardization into JPEG-LS"* In: IEEETIP: IEEE Transactions on Image Processing, 1998

[GYM93] B. Guenter, H. Yun, R. Mersereau. Motion compensated compression of computer animation frames. In James T. Kajiya, editor: *Computer Graphics (SIGGRAPH '93 Proceedings),* volume 27, pages 297 – 304, August 1993.

[G91] D. Gall, "*MPEG: A video compression standard for multimedia applications*, " Commun. ACM, vol. 34, pp. 46--58, Apr. 1991.

[Mpeg98] MPEG1 - ISO/IEC 11172-2, *"Information Technology – Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1,5 Mbit/s – Video"*, Geneva, 1993

[Mpeg94] MPEG2 - ISO/IEC FTC1/SC29/WG11 N0702 Rev, *"Information Technology – Generic Coding of Moving Pictures and Associated Audio, Recommendation H.262"* Draft International Standard, Paris, 25 March 1994.

[SDL*05] V. Sundstedt, K. Debattista, P. Longhurst, A. Chalmers and T. Troscianko. *"Visual Attention for Efficient High-Fidelity Graphics."* In: Spring Conference on Computer Graphics (SCCG 2005), May 2005.

[YPG01]H. Yee, S. Pattanaik and D. Greenberg. "Spatiotemporal sensitivity and Visual Attention for efficient rendering of dynamic Environments", In *ACM Transactions on Computer Graphics*, Vol. 20, No. 1, 39-65, 2001.

[S05] Veronica Sundstet – Animation created for [SDL*05] – copyright 2005, image used with permission.