Motion Compensated Compression of Computer Animation Frames *

Brian K. Guenter †, Hee Cheol Yun, and Russell M. Mersereau ‡

Abstract

This paper presents a new lossless compression algorithm for computer animation image sequences. The algorithm uses transformation information available in the animation script and floating point depth and object number information stored at each pixel to perform highly accurate motion prediction with very low computation. The geometric data, i.e., the depth and object number, is very efficiently compressed using motion prediction and a new technique called direction coding, typically to 1 to 2 bits per pixel. The geometric data is also useful in z-buffer image compositing and this new compression algorithm offers a very low storage overhead method for saving the information needed for z-buffer image compositing. The overall compression ratio of the new algorithm, including the geometric data overhead, is compared to conventional spatial linear prediction compression and is shown to be consistently better, by a factor of 1.4 or more, even with large frame-to-frame motion.

CR Categories: I.4.2[compression(coding)]exact coding.
Additional keywords: compression,computer animation,computer graphics, motion prediction

1 Introduction

With the increasing popularity and falling cost of computer animation comes a new problem: storing the enormous data files which even short computer animation sequences require. Five minutes of NTSC resolution computer animation takes up approximately 8.5 gigabytes of storage; film resolution takes many times more. This amount of data cannot be economically stored on line in high speed secondary storage devices. Animation image files are typically stored off-line on removable media.

An alternative to using off-line storage is to compress the image data and store it on-line. This is very desirable for sequence editing and image manipulation, for example. For high image quality only lossless compression is acceptable; images can then be exactly reconstructed from their compressed representation. Errors do not accumulate if images are combined or manipulated and run through the compression decompression cycle several times.

Much more information is available to a compression algorithm for computer animation than is the case for live action video. However, surprisingly little work has been done on exploiting the information in a computer animation script to improve image compression efficiency. Previous work such as that described in [1] and [5] is actually image based compression although the application is to computer animation.

The lossless compression algorithm for computer animation to be described in this paper combines elements of both motion prediction and spatial linear prediction compression techniques, using each when most appropriate. The new compression algorithm uses transformation information in the animation script to perform essentially perfect image space motion prediction with very low computation. This is a major advantage of the new algorithm because motion prediction with subpixel accuracy based only the information present in the image sequence is computationally expensive [6][3]. Poor quality motion prediction increases the motion prediction error which reduces the maximum achievable compression ratio.

One of the most effective lossless image compression techniques is DPCM followed by entropy coding [4][8]. For typical live action video sequences the best compression achievable using this method is usually less than 2 to 1 [7]. The best computer generated images are nearly indistinguishable from real images so we can expect that good synthetic images will not compress any better than live action video.

For scenes with fairly rapid camera and object motion the compression ratio we have achieved with our new motion prediction compression is approximately 1.5 times that of spatial linear prediction compression techniques - about 3 to 1 compression with the new technique as opposed to 2 to 1 compression with DPCM. As camera and object motion decrease the compression ratio of the new technique steadily increases while spatial prediction compression remains constant at roughly 2 to 1.

Extra geometric information, the object number and the depth at each pixel, is stored in each frame to perform motion prediction. The geometric information is compressed very efficiently in our new algorithm, typically to 1 or 2 bits per pixel. For z-buffer compositing applications [2] this is another advantage of the new algorithm, because the depth information needed for z-buffer compositing is stored in very little space.

We assume the animation script contains a homogeneous matrix transformation for every object in every frame. The matrix transforms the object from the model space coordinate frame into the screen space coordinate frame. The transformation matrices are stored in an auxiliary file along with the compressed image data and constitute part of the overhead of the new compression algorithm. This limits the current implementation to rigid body motion but this is not an intrinsic limitation of the algorithm. Non-rigid body motion can be accommodated by storing appropriate transformation information, such as free form deformation

*This work was supported by the National Science Foundation under grant MIP-9205853

†Computer Animation Laboratory, GVU center, College of Computing, Georgia Institute of Technology, Atlanta GA 30332, E-mail: brian.guenter@cc.gatech.edu

‡Digital Signal Processing Labratory, School of Electrical Engineering, Georgia Institute of Technology, Atlanta GA 30332, E-mail: yun@eedsp.gatech.edu, rmm@eedsp.gatech.edu

mesh points for example [9], in the animation script.

The current implementation assumes that objects are represented as polygonal surfaces. Algorithms exist for converting many different surface representations to approximating polygonal surfaces. Many commercial image synthesis programs perform this conversion internally so the limitation to a polygonal representation is not unduly restrictive.

The geometric data has special properties we exploit to improve compression. As a consequence the coder is split into two parts: a geometrical data coder and a color data coder. General notation used throughout the paper is presented in Section 2.

Section 3 of the paper presents block diagrams of the algorithm. Section 4 describes the geometrical data coding algorithm. Section 5 describes the color data coding algorithm. Animation test results are presented in section 6 and conclusions and suggestions for further research are presented in section 7.

2 Notations and Data structure

In this paper the frame number, which is used to identify the specific frame, is expressed as a superscript. A subscript represents the object number when it is expressed as a single value and the spatial location when it is expressed as a pair of values. If the subscripts are omitted, that symbol represents the whole set of the corresponding data for that frame.

The data structure of a frame is divided into two parts. The first part is the set of 4×4 homogeneous matrices for all the objects $\{ T^i_j, j = 0..N_o - 1 \}$ by which the point in the model object space is transformed to the screen space. The other part is the 2-dimensional array of the data $P^i_{m,n}, m = 0..N_x - 1, n = 0..N_y - 1$, where N_o represents the number of objects and N_x and N_y represent the number of pixels in each direction. Each pixel datum $P^i_{m,n}$ is composed of the object number $N^i_{m,n}$, depth $Z^i_{m,n}$ and colors $C^i_{m,n}$ of the pixel at the spatial location (m, n). For example, the point in the i-th frame $(m, n, Z^i_{m,n})^T$ is transformed to the point $(x^j, y^j, z^j)^T$ in the *j-th* frame as follows:

$$T^j_k (T^i_k)^{-1} \begin{pmatrix} m \\ n \\ Z^i_{m,n} \\ 1 \end{pmatrix} = \alpha \begin{pmatrix} x^j \\ y^j \\ z^j \\ 1 \end{pmatrix} \qquad (1)$$

where $k = N^i_{m,n}$.

As mentioned above, the symbols without the subscripts represent the whole set of data for the frame. For example T^i stands for the set of matrices and N^i, Z^i, C^i represent the whole two dimensional array, also called a field, containing the object number, depth and color values of the *i-th* frame, respectively. The object number N^i and the depth Z^i are collectively called the geometrical data field. The color field C^i represents the R,G,B color fields, but sometimes can be used for one specific color field.

The object number and color values are represented as integers, but the depth is a real number. In our implementation, each of the RGB color values is usually represented by 8 bits/pixel (256 levels), and the depth is double precision floating point. Since the compression efficiency of the geometrical data is highly dependent on the accuracy of the calculation, the double precision representation is preferred. The required number of bits for the object number depends on the total number of objects.

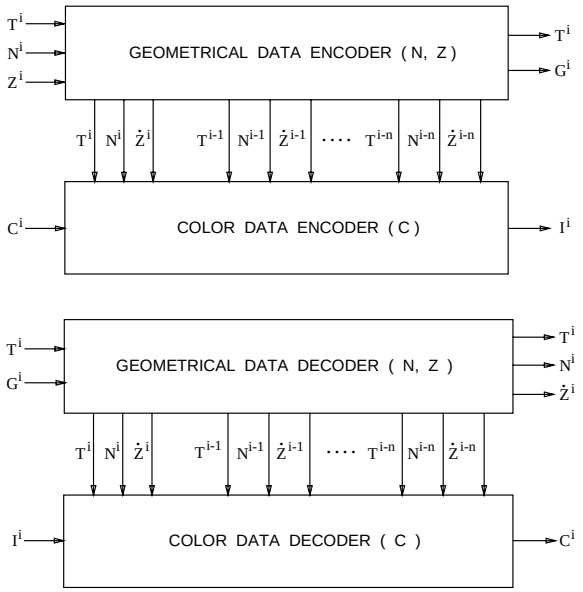

Figure 1: *n-th* Order Frame Encoder and Decoder

3 System Block diagrams

The heart of the coding scheme uses a linear predictive coding algorithm (DPCM) [4]. Since there exists substantial correlation between successive frames in computer animation as well as in real-life video, good compression gain can be achieved by these predictive schemes. Since both the object number and depth fields are needed to compute the motion trajectory of each pixel and these are encoded together into one data stream G^i, the whole system is divided into a geometrical data coding block for N^i, Z^i and a color data coding block for C^i as shown in Figs 1.

The object number N^i and color data C^i are coded losslessly, but the depth Z^i is allowed to contain error within a specified limit to achieve a high compression gain, because Z^i requires a relatively larger number of bits (64 bits/pixel for a double precision representation) than N^i, C^i.

The DPCM system requires storage for several frames determined by the order of the predictor. The geometrical data coding block stores the object number fields N^i and the depth fields $\dot{Z}^i$ of previous frames. Since only the decoded values are available for the depth field in the decoder, the geometrical data encoder uses the decoded depth field $\dot{Z}^i$ instead of the original depth field Z^i for correct reconstruction from the encoded data. The stored geometrical data $N^i, \dot{Z}^i$ are provided to the color data coding block to predict the color data in other frames by motion prediction.

4 Geometrical Data Coding

Figs 2 show the block diagrams of the encoder and decoder for geometrical data. The principle behind the geometrical data coding is that the geometrical data of the current frame is predicted from several previous frames which are compared to the current original frame pixel-by-pixel. Each pixel $P^i_{m,n}$ is classified as matched if $N^i_{m,n}$, $Z^i_{m,n}$ of the current frame are the same as $\tilde{N}^i_{m,n}$, $\tilde{Z}^i_{m,n}$ of the predicted frame, and unmatched otherwise. Since the object number and depth for the matched pixels can be recovered from the

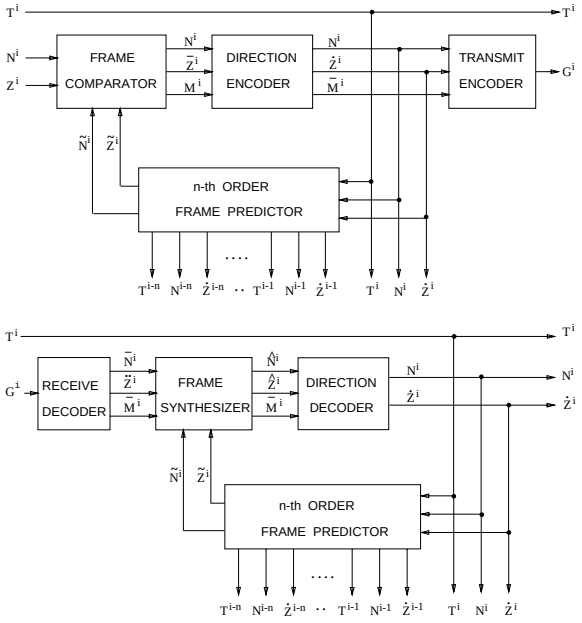

Figure 2: Geometrical Data Encoder and Decoder

predicted frame in the decoder, the only information that needs to be transmitted are the matching status field M^i which records whether or not each pixel is matched, and the complete geometrical data for the unmatched pixels. Unmatched pixels occur mainly in recently uncovered regions which cannot be predicted from previous frames, or from highly curved regions that are difficult to predict.

For the unmatched regions the geometrical data can be coded effectively by exploiting the spatial correlations between pixels, because pixels which belong to the same planar polygon satisfy the same plane equation. An algorithm called direction coding is proposed and described later. With direction coding, the unmatched pixels are classified into direction matched pixels and totally unmatched pixels. This matching information replaces $M^i_{m,n}$ at the unmatched pixel and this modified matching status field is $\bar{M}^i$. After the direction coding, since the majority of the frame is matched, the entropy of $\bar{M}^i$ is very small. Thus $\bar{M}^i$ can be compressed effectively by entropy coding or run-length coding and the original geometrical data $N^i_{m,n}$, $Z^i_{m,n}$ are transmitted in uncompressed form only for the totally unmatched pixels.

At the receive decoder, the matching status field $\bar{M}^i_{m,n}$, and the geometrical data $N^i_{m,n}$,$Z^i_{m,n}$ for totally unmatched pixels are obtained. For the matched pixels which can be identified by the $\bar{M}^i$, the object number $N^i_{m,n}$ and the depth $\tilde{Z}^i_{m,n}$ are copied from the predicted frame. Then for the unmatched pixels the $N^i_{m,n}$, $\acute{Z}^i_{m,n}$ are recovered by the direction decoder and the complete recovered frame data is fed into the frame predictor for the next frame prediction.

4.1 Frame Predictor

The set of four pixels $P^i_{m,n}$, $P^i_{m,n+1}$, $P^i_{m+1,n+1}$, $P^i_{m+1,n}$ is defined as a pixel square $S^i_{m,n}$. Each pixel square can be classified into one of three categories. The first is the plane pixel square where all four corner pixels come from the same planar polygon and make a planar square. The second is the adjacent polygon pixel square where four pixels are from the

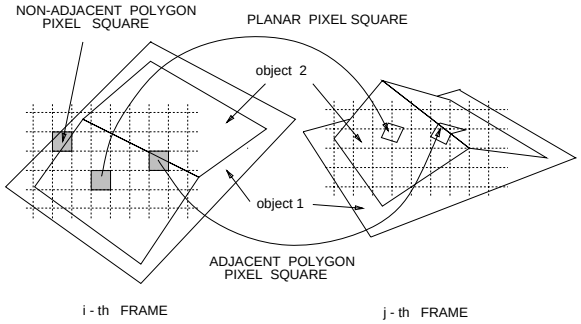

Figure 3: Transform of Pixel Squares between Frames

two adjacent polygons that share an edge that intersects the pixel square. The third is a non-adjacent polygon pixel square where the polygon boundary is across the pixel square but the polygons do not share an edge. These three cases are illustrated in Fig. 3.

The planar pixel square can be easily transformed by Eq. 1 and rendered into other frames. For the adjacent polygon pixel square, if the plane equations of the two polygons can be obtained by exploiting the neighboring pixel squares, then the pixel square can be partitioned into two polygons and each polygon can be transformed and rendered onto other frames in the same way. There is not enough information to make correct partitions for the non-adjacent polygon pixel squares, then these cannot be used to predict other frames. The pixel square $S^i_{m,n}$ can be defined as a planar pixel square if the four pixels satisfy the following plane conditions.

$$N^i_{m,n} = N^i_{m,n+1} = N^i_{m+1,n+1} = N^i_{m+1,n} \tag{2}$$

$$|Z^i_{m,n} + Z^i_{m+1,n+1} - Z^i_{m,n+1} - Z^i_{m+1,n}| < \epsilon \tag{3}$$

In Eq. 3 the inequality is used to deal with the error due to the limited precision of computation and the small number ϵ is determined as the allowable error for the depth of the pixels which are from the same planar polygon.

In some cases, the above two conditions are not enough to determine whether the pixel square is planar or not. For example, the pixels lying across the boundary of two separate polygons which are parallel to each other might satisfy these two conditions. There are several ways of reducing the possibility of an incorrect classification of a planar pixel square. One way is to add the following conditions, which test the relations between the depths of the surrounding pixels. If the following plane conditions are satisfied with regard to at least one corner of the pixel square, $S^i_{m,n}$ can be considered to be a plane pixel square.

$$|2\,Z^i_{l,k} - Z^i_{l-1,k} - Z^i_{l+1,k}| < \epsilon$$

$$|2\,Z^i_{l,k} - Z^i_{l,k-1} - Z^i_{l,k+1}| < \epsilon \tag{4}$$

$$\text{for } l = m, m+1 \text{ and } k = n, n+1.$$

For the non-planar pixel square which does not satisfy the above plane conditions, if some two pixel squares around it are planar and if the intersection of those two planes are found to be across the pixel square by solving the plane equations of those two planes, then this pixel square is an adjacent polygon pixel square that can be divided into two polygons and transformed into other frames. One way to find the two plane pixel squares is to test the above plane

conditions for each pair of pixel squares which are on the opposite sides of the current pixel square.

The geometrical data for most of a frame can be computed by transforming all the planar and adjacent polygon pixel squares of the previous frame into the current frame and rendering the transformed polygons on the frame buffer of geometrical data using the z-buffer algorithm.

Since the current frame does not change much from the previous frame, the transformed polygon of one pixel square is small and covers only a few pixels. Under this assumption, there are several effective techniques for geometrical data rendering. One simple method is to find the bounding box of the transformed pixel square and test whether each pixel point inside the bounding box is inside the polygon or not. For each inside pixel point, the depth of that point can be computed from the plane equation of the transformed polygon for the z-buffer rendering process. The back-face removal step might be applied before rendering.

There is one special case where the viewpoint and object are not moving. In this case the transform matrices of the current and previous frames are the same and the whole transform matrix $T_k^j(T_k^i)^{-1}$ will be the identity matrix in Eq. 1. For these pixels, the above complicated steps are not necessary and the only thing to do is simply to apply the depth of the previous frame to the z-buffer algorithm at the same pixel location. All these pixels are classified as matched pixels.

Due to occlusion some parts of the current frame may not be predictable from the previous frame. The percentage of predictable pixels can be increased by using higher order prediction. In the n-th order case, the previous n frames are transformed and rendered on the same frame buffers of object number and depth.

The frame predictor used in both the encoder and decoder have identical frame buffers for a object number and depth for higher order prediction. The encoder and decoder should store the same frame data in both predictors so that the predicted frames in both blocks will be the same.

4.2 Frame Comparator

The frame comparator compares the input frame data to the predicted frame data on a pixel-by-pixel basis and records the result in the matching status field M^i. If a pixel $P^i_{m,n}$ of the current frame and $\tilde{P}^i_{m,n}$ of the predicted frame satisfy the following conditions, then the pixel is considered to be predictable from the previous frames and said to be a matched pixel.

$$N^i_{m,n} = \tilde{N}^i_{m,n} \tag{5}$$

$$|Z^i_{m,n} - \tilde{Z}^i_{m,n}| < \epsilon \tag{6}$$

In Eq. 6, an inequality is used for the same reason as in Eq. 3. If a pixel $P^i_{m,n}$ is found to be matched, $M^i_{m,n}$ is set to 1 and otherwise set to 0. Because of the similarity between the adjacent frames, most of M^i will be 1. For matched pixels, the depth $Z^i_{m,n}$ of the input frame is replaced by the predicted depth $\tilde{Z}^i_{m,n}$ to guarantee the consistency of data between the encoder and decoder, because only $\tilde{Z}^i_{m,n}$ will be available in the decoder. By Eq. 6 the accuracy of the new depth $\tilde{Z}^i_{m,n}$ is guaranteed to be within ϵ. This modified depth field is $\bar{Z}^i$ and will be given to the direction coder along with N^i and M^i.

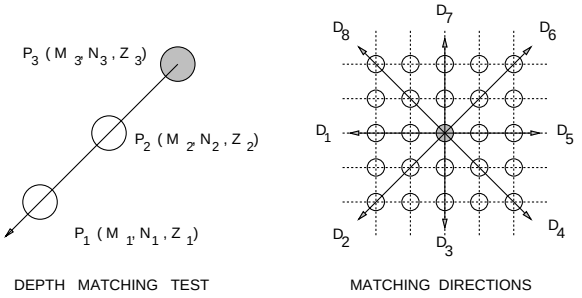

Figure 4: Determination of Direction Matching

4.3 Direction Encoder

Unmatched regions are usually from recently uncovered regions or highly curved regions that cannot be predicted. Since the pixels in those regions might come from some plane polygons or might be on the extension of a plane from a surrounding matched regions, a spatial prediction technique exploiting the plane relationship between neighboring pixels can be used to code these pixels. One such method is to find the matching direction in which two neighbor pixels lying on a straight line match the object number and depth of the current pixel and record the direction value by overwriting the $M^i_{m,n}$ which was originally zero. Since only the reconstructed data are available in the decoder, those two pixels should be pixels that have been already coded. Therefore it should be checked whether the matching status values of those two pixels is still zero. The matching conditions for a direction at P_3 in Fig. 4 are described as follows :

$$M_1 \neq 0 \text{ and } M_2 \neq 0 \tag{7}$$

$$N_1 = N_2 = N_3 \tag{8}$$

$$|Z_3 - \acute{Z}_3| < \epsilon \tag{9}$$

where $\acute{Z}_3$ is the spatially predicted depth of the pixel P_3 in the specified direction as in Eq. 10.

$$\acute{Z}_3 = 2Z_2 - Z_1 \tag{10}$$

These conditions are tested for eight directions from direction D_1 to D_8 as in Fig. 4. The first matched direction becomes the matching direction of the pixel and the corresponding direction value, which is defined as $D_i = i + 1$ in Fig. 4, is assigned to $M^i_{m,n}$ which was originally zero. The depth of the current pixel is replaced by the predicted value in the matched direction as in Eq. 10 for consistency of the data in the encoder and decoder. If there is no matching direction, the number 10 is assigned, corresponding to a totally unmatched pixel. After direction coding, the frame data will be:

$$\bar{M}^i_{m,n} = \begin{cases} 1 & \text{matched} \\ 2..9 & \text{direction matched} \\ 10 & \text{totally unmatched} \end{cases} \tag{11}$$

$$\dot{Z}^i_{m,n} = \begin{cases} \tilde{Z}^i_{m,n} & \text{matched} \\ \acute{Z}^i_{m,n} & \text{direction matched} \\ Z^i_{m,n} & \text{totally unmatched} \end{cases} \tag{12}$$

N^i remains unchanged because the object number is losslessly coded.

Fig. 5 shows an example of direction coding where the region inside the polygon is originally unmatched. The encoding is performed from left to right and from bottom to top

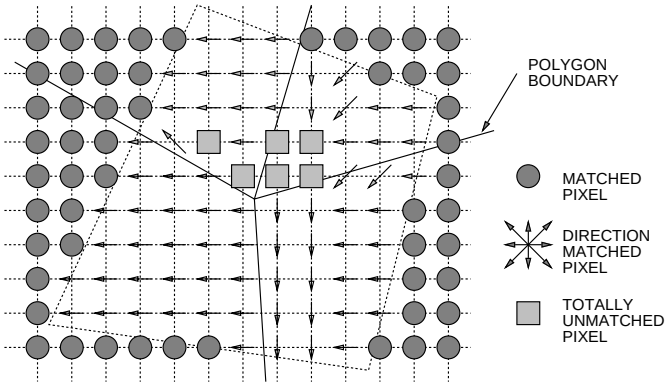

Figure 5: Example of Direction Encoding. The unmatched region inside the dashed polygon is coded by direction coding. Coding is performed from left to right and from bottom to top.

and the arrows represent the direction of matching. This illustrates that the number of totally unmatched pixels is very small and the matching directions are mostly 2 because direction 2 is the first test direction in Fig. 4. Since the geometrical data for the matched and direction matched pixels are predictable, the data to be transmitted are the matching status field $\bar{M}^i$ which has very low entropy and the $\{N^i_{m,n}, Z^i_{m,n}\}$'s for a few totally unmatched pixels.

4.4 Frame Synthesizer

Since the matched pixels can be identified from the matching status field $\bar{M}^i$ decoded in the receive decoder, the frame synthesizer can recover the geometrical data for all the matched pixels by copying the data from the predicted frame. Then the geometrical data $\hat{N}^i_{m,n}$, $\hat{Z}^i_{m,n}$ for the matched pixels and totally unmatched pixels are correctly recovered whereas those for direction matched pixels remain undetermined:

$$\hat{N}^i_{m,n} = \begin{cases} \bar{N}^i_{m,n} = N^i_{m,n} & \text{if } \bar{M}^i_{m,n} = 10 \\ \tilde{N}^i_{m,n} = N^i_{m,n} & \text{if } \bar{M}^i_{m,n} = 1 \\ \text{undetermined} & \text{if } \bar{M}^i_{m,n} = 2..9 \end{cases} \quad (13)$$

$$\hat{Z}^i_{m,n} = \begin{cases} \ddot{Z}^i_{m,n} = Z^i_{m,n} & \text{if } \bar{M}^i_{m,n} = 10 \\ \tilde{Z}^i_{m,n} & \text{if } \bar{M}^i_{m,n} = 1 \\ \text{undetermined} & \text{if } \bar{M}^i_{m,n} = 2..9 \end{cases} \quad (14)$$

4.5 Direction Decoder

For the direction matched pixels, the object number $N^i_{m,n}$ is recovered by copying the object number of the pixel which is located in the matching direction and the depth $\dot{Z}^i_{m,n}$ becomes the predicted value $\acute{Z}^i_{m,n}$ from the two pixels located in the matching direction as in Eq. 10. $N^i_{m,n}$ are recovered correctly for all the pixels and the depth field $\dot{Z}^i$ is the same as $\dot{Z}^i$ in Eq. 12. The accuracies of $\tilde{Z}^i_{m,n}$ for the matched pixel and $\acute{Z}^i_{m,n}$ for the direction matched pixel are guaranteed by Eq. 6 and Eq. 9, respectively. These decoded data are fed into the predictor and are the same for both the encoder and decoder to make the same predictions as in Fig. 2.

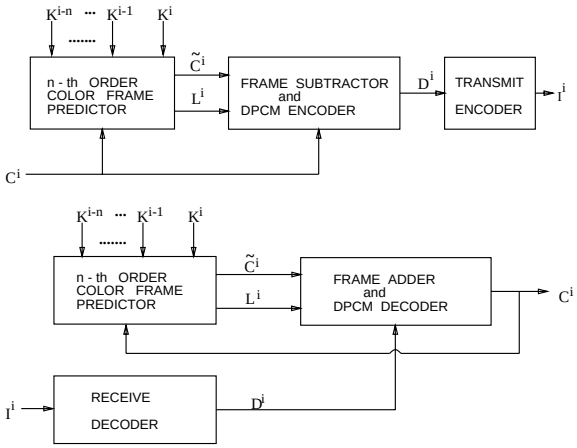

Figure 6: Color Data Encoder and Decoder, where $K^i = (T^i N^i \dot{Z}^i)$

5 Color Data Coding

Since the (T, N, Z)'s from the geometrical data coding block can be used to compute the locations of a current frame pixel on the previous frames and the previous color data frames are stored, the color of the pixel can be predicted by estimating the color at the transformed locations in the previous frames. The error image between the original and the predicted frame, also called the residual image R^i, has a relatively low entropy compared to the original. The pixels are classified into two classes, matched and unmatched pixels, based on whether the locations in the old frames are traceable or not. Since this residual image still has some spatial correlation, spatial linear prediction coding (DPCM) can be applied to reduce the entropy. This DPCM coded residual image is called a differential residual D^i and entropy coding using Huffman coding or arithmetic coding, can compress the differential residual losslessly. In decoder after the residual R^i is recovered by DPCM decoding, the original color data $C^i_{m,n}$ is obtained as the sum of the predicted value $\tilde{C}^i_{m,n}$ and the residual $R^i_{m,n}$ for the matched pixel. The color $C^i_{m,n}$ of the unmatched pixel is the same as residual $R^i_{m,n}$.

5.1 Color Frame Predictor

A pixel of one frame can be mapped to another frame by the transformation in Eq. 1 and if the transformed point satisfies the following condition, it is considered to be the same point as the current pixel and the pixel is said to be matched. If the pixel $P^i_{m,n}$ is transformed into the j-th frame and the transformed point is inside the pixel square $S^j_{p,q}$, the matching condition is as follows:

$$Z^j_{min} \le Z^j_{trans} \le Z^j_{max} \quad (15)$$

where the Z^j_{trans} is the depth of the transformed point and Z^j_{min} , Z^j_{max} are the minimum and maximum depth of the four corner pixels of $S^j_{p,q}$, respectively. Since several previous frames of geometrical and color data are available to deal with the occlusion problem, the transformed point in the nearest frame which satisfies the above condition is used to predict the color data of the current pixel.

L^i records the matching status values for color data which are zeros for the unmatched and ones for the matched pixels. Since generally the transformed point is not the pixel point, an interpolation is necessary for the computation of the color.

5.2 Subtractor and DPCM Encoder

Colors of matched regions in the residual R^i are residual values generated by sutracting the predicticed colors from the original colors whereas the colors in the unmatched regions remain unchanged as follows:

$$R^i_{m,n} = \begin{cases} C^i_{m,n} & \text{if } L^i_{m,n} = 0 \\ C^i_{m,n} - \tilde{C}^i_{m,n} & \text{if } L^i_{m,n} = 1 \end{cases} \quad (16)$$

Unmatched regions are mostly from recently uncovered regions and the entropy of such unmatched regions can be reduced by spatial linear predictive coding (DPCM). The residual image in the matched region has low entropy caused by changes of illumination, by the movement of objects, viewpoints or light sources between frames. Since this kind of error has relatively slow spatial variation, DPCM can be effectively applied also to the matched regions of the residual image.

These two steps, frame subtraction and DPCM, can be implemented with a combined operation as in Fig. 6. Since the matched and unmatched regions have different kind of data as explained above, each region should be coded independently. Usually 2-D DPCM uses three left and lower neighboring pixels $\acute{C}_{p,q}$ to predict the current pixel $C^i_{m,n}$. If the current pixel is an unmatched pixel, $\acute{C}^i_{p,q}$ should be the original color $C^i_{p,q}$. If the current pixel is a matched pixel, $\acute{C}^i_{p,q}$ should be the residual value. Since the residual value is not available for an unmatched pixel, the $\acute{C}^i_{p,q}$ is set to zero when $L^i_{p,q} = 0$ as follows:

$$\acute{C}^i_{p,q} = \begin{cases} C^i_{p,q} & \text{if } L^i_{m,n} = 0 \\ L^i_{p,q}(C^i_{p,q} - \tilde{C}^i_{p,q}) & \text{if } L^i_{m,n} = 1 \end{cases} \quad (17)$$

The spatially predicted value $\dot{C}^i_{m,n}$ is the integer part of a linear combination of those as follows:

$$\dot{C}^i_{m,n} = \text{int}(\alpha \acute{C}^i_{m-1,n} + \beta \acute{C}^i_{m-1,n-1} + \gamma \acute{C}^i_{m,n-1}) \quad (18)$$

In this paper, the prediction coefficients α, β, γ are selected to be 0.75,-0.5,0.75 respectively.

Then the differential residual image D^i is obtained by subtracting the spatially predicted value $\dot{C}^i_{m,n}$ from the residual value as follows:

$$D^i_{m,n} = \begin{cases} C^i_{m,n} - \dot{C}^i_{m,n} & \text{if } L^i_{m,n} = 0 \\ C^i_{m,n} - \tilde{C}^i_{m,n} - \dot{C}^i_{m,n} & \text{if } L^i_{m,n} = 1 \end{cases} \quad (19)$$

The differential residual D^i, which has very small entropy, can be compressed losslessly by an entropy coding technique.

5.3 Adder and DPCM decoder

The $L^i_{m,n}$ and $\tilde{C}^i_{m,n}$ in the predictor of the decoder are the same as those in the encoder. The original color field C^i is recovered by the combined step of DPCM decoding and frame addition as follows:

$$C^i_{m,n} = \begin{cases} D^i_{m,n} + \dot{C}^i_{m,n} & \text{if } L^i_{m,n} = 0 \\ D^i_{m,n} + \tilde{C}^i_{m,n} + \dot{C}^i_{m,n} & \text{if } L^i_{m,n} = 1 \end{cases} \quad (20)$$

where the spatial prediction $\dot{C}^i$ is the same as in the encoder. The recovered color C^i is fed back to the predictor for the prediction of the next frames.

6 Test Results

31 animation frames were generated to test the proposed compression algorithm. Each frame is composed of 7 objects and various kinds of textures were mapped by solid texture mapping. Through the whole 31 frames the ball bounces back and forth between the two wood blocks. In the first 11 frames the viewpoint does not change and in the next 10 frames the view point moves approximately 5 degrees/frame. During the last 10 frames zooming is performed. Plate 2,3, and 4 show the frames for above three cases.

In the case of Plate 2, since most of the objects are not moving and the view point is fixed, all regions except the rolling ball are matched regions. Pixel points in the current frame are transformed exactly to the same pixel points of other frames, then there are no errors due to bilinear color interpolation and since even the colors on the edges of stationary objects are predictable, the residual will have extremely small entropy which results in very high compression gain.

Plate 3 is a more general case in which both an object and the viewpoint are moving.There are mainly three kinds of residual errors. First, the recently uncovered regions are the major error regions which can be compressed only by DPCM. Second, in matched regions the changes in illuminations on the object surfaces causes residual errors. Usually illumination changes are due to the changes of specular reflection which varies with the movement of the view point or the object itself. Generally these errors change slowly and can be lowered by DPCM. Third, since the object boundaries are often unmatched regions which do not satisfy Eq. 15, the residual errors are large there. And even in the case where the pixels on the object boundary are matched, relatively large errors due to the color interpolation occur because the color data of these pixels were generated by antialiasing. In Plate 4, since with zooming there are no recently uncovered regions and the spatial frequency is decreasing, the entropy of the residual signal will be smaller than that in Plate 3.

The second order compression algorithm was implemented and tested on this test sequence. Fig. 7 shows the entropies of the original pictures and the differential residual images by 2-D DPCM and motion prediction algorithm. Table 1 illustrates the entropies of several frames. Plates 5 and 6 show the differential residual images of the RED component by linear predictive coding (DPCM) and the new motion prediction, respectively. As explained above, the major errors in Plate 6 are on recently uncovered regions and along the object boundaries which are often unmatched regions. The biggest differences between DPCM and motion prediction occurred on the wood texture which has relatively higher spatial frequency than any other regions. In these high spatial frequency regions, motion prediction shows much higher performance than spatial linear predictive coding. Through the whole 31 frames, the entropy of the original frame is around 20.7 bits/pixel. In the first 11 frames where the viewpoint is fixed, only about 1 bits/pixel is required for the motion prediction technique, except for the first frame which cannot be motion predicted and is coded only by DPCM. This contrasts with DPCM which needs around 11.4 bits/pixel. The necessary bits/pixel for the next 10 frames in which both the viewpoint and object are moving is around 8.3 bits/pixel which is about 3 bits/pixel gain over DPCM.

	frame 5	frame 15	frame 25
$N_{ORIGINAL}$	20.88	20.78	20.58
N_{DPCM}	11.35	11.23	10.59
N_{GEOM}	0.32	1.06	0.93
N_{RGB}	0.67	7.20	6.19
N_{MOTION}	0.99	8.26	7.12

Table 1: Entropies of original image and residuals by DPCM and Motion Prediction (unit : bits/pixel)

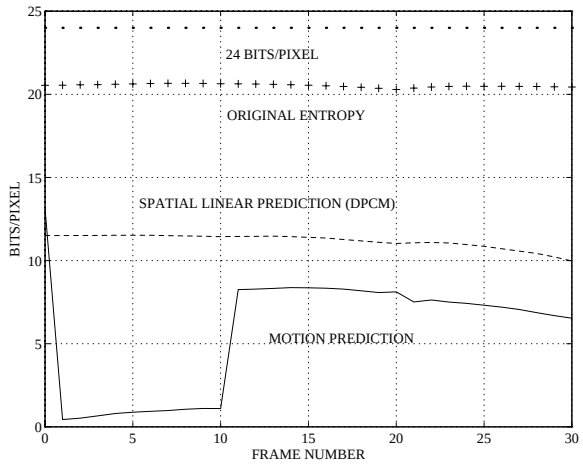

Figure 7: Entropies of the Bouncing Ball Sequence. The data for motion prediction are the sum of compressed geometrical data and entropy of residual.

In the next 10 frames, all entropies are decreasing with zooming as expected above, but the motion prediction algorithm still outperforms DPCM by about 3.5 bits/pixel.

7 Conclusion

The motion prediction compression algorithm for computer animation image sequences presented here consistently outperforms spatial linear prediction. This is true even though the new algorithm encodes double precision depth and integer object number information for a total of 96 bits/pixel including RGB data while the spatial compression algorithm does not. The depth and object number information are useful in their own right for performing z-buffer image compositing. The compression ratio achieved by the new algorithm was 1.4 times or more greater than that achieved with spatial linear prediction even for scenes with rapid changes in camera view point and substantial changes in object occlusion relationships. The overall compression ratio achieved with the new algorithm for image sequences with significant object and viewpoint motion was approximately 3 to 1.

There is still scope for improvement in the algorithm. We have noticed significant remaining correlation in the residual images. Long strings of zero residual values are punctuated by short bursts of plus and minus one residual values. Additionally the geometric overhead can be significantly reduced by storing the polygonal surface data and re-rendering it with the z- buffer algorithm, a process which should be no more time consuming than re-rendering pixel squares as in the current implementation. This makes the auxiliary data file more complex but for animation sequences more than a few seconds long the geometric data overhead should drop to .1 bit/pixel or less. We are currently implementing this extension.

The current implementation of the algorithm is limited to objects represented as polygonal surfaces. As discussed in the introduction this is not an undue restriction since efficient algorithms exist for approximating many different surface types with polygonal surfaces. However it would be an interesting research project to extend the geometric coder to other surface types so that an exact, rather than an approximating polygonal, surface representation could be used. This extension would require modifying both the motion prediction stage and the direction coding stage to compute surface equations directly from the depth information stored in the image.

8 Acknowledgments

The authors acknowledge help from Ragnar Jonsson, Stephen A. Martucci, Wayne Wooten and Lonnie D. Harvel.

References

[1] Denber, Michael J. and Turner, Paul M. A Differential Compiler for Computer Animation. Proceedings of SIGGRAPH '86 (Dallas, Taxas, August 18–22,1986). In *Computer Graphics* 20,4 (August 1986), 21–27.

[2] Duff, Tom. Compositing 3-D Rendered Images. Proceedings of SIGGRAPH '85 (San Francisco, California, July 22–26,1985). In *Computer Graphics* 19, 3(July 1985), 41–44.

[3] Jain, Anil K. *Fundamentals of Digital Image Processing.* Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

[4] Jayant, Nuggehally S. and Noll, Peter. *Digital Coding of Waveforms.* Prentice-Hall, Englewood Cliffs, New Jersey, 1984.

[5] Jones, Stephen C. and Moorhead II, Robert J. Hardware-specific Image Compression Techniques for the Animation of CFD data. Proceedings of SPIE - International Society for Optical Engineering, 1668 (San Jose, California, February 10–11,1992), 141–146.

[6] Lim, Jae S. *Two-Dimensional Signal and Image Processing.*, Prentice-Hall, Englewood Cliffs, New Jersey, 1990.

[7] Martucci, Stephen A. Reversible Compression of HDTV Images Using Median Adaptive Prediction and Arithmetic Coding. Proceedings of IEEE ISCAS, 2(New Orleans, Louisiana, May 1–3,1990),1310–1313.

[8] Melnychuck, Paul W. and Rabbani, Majid. Survey of Lossless Image Coding Techniques. Proceedings of SPIE - International Society for Optical Engineering, 1075 (Los Angeles, California, January 17–20, 1989), 92–100.

[9] Sederberg, Thomas W. and Parry, Scott R. Free-Form Deformation of Solid Geometric Models. Proceedings of SIGGRAPH '86 (Dallas, Taxas,August 18–22,1986). In *Computer Graphics* 20,4 (August 1986), 151–160.

[10] Witten, Ian H., Neal, Radford M., and Cleary, John G. Arithmetic Coding for Data Compression. In *Communications of the ACM* 30,6 (June 1987), 520–540.

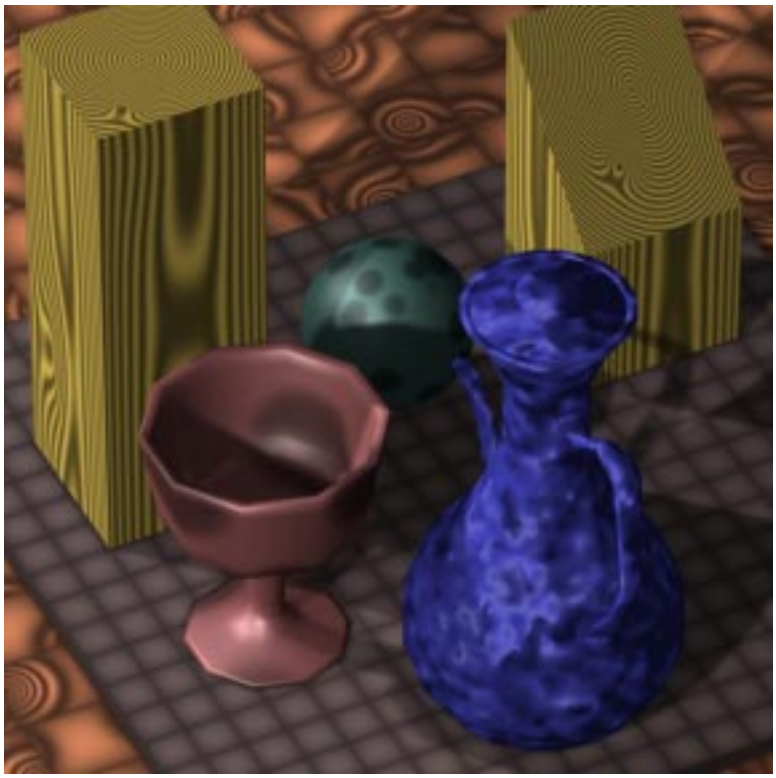

Plate 1: Frame 5 of the test sequence

Plate 2: Frame 0,9

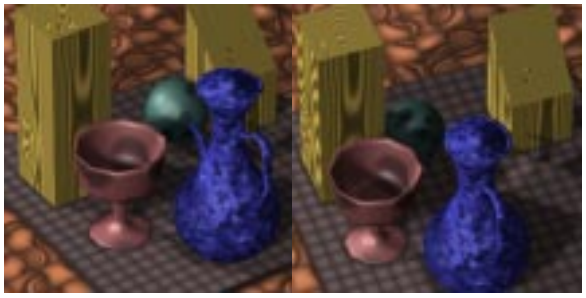

Plate 3: Frame 10,19

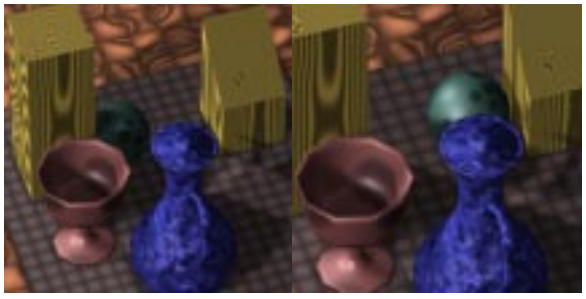

Plate 4: Frame 20,29

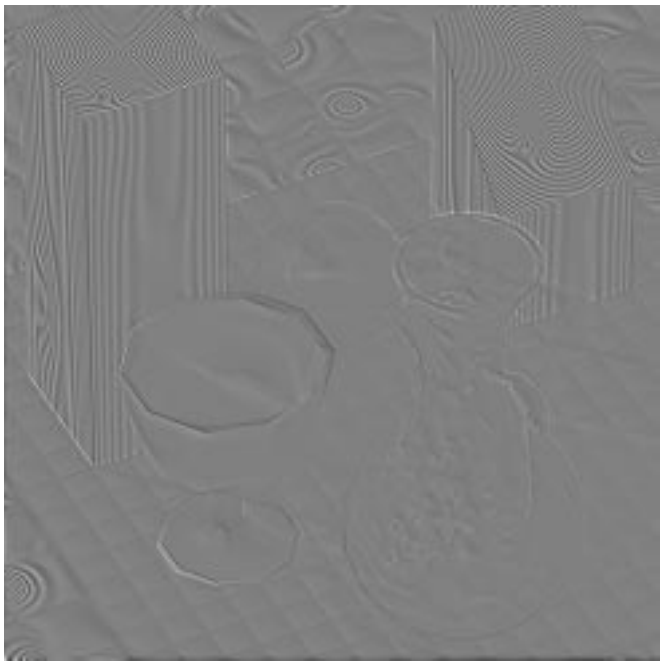

Plate 5: The differential image of frame 5 by DPCM (512×512).

Plate 6: The differential residual image of frame 5 by motion compensation (512×512).