Accelerated MPEG Compression of Dynamic Polygonal Scenes

Dan S. Wallach
dwallach@cs.princeton.edu

Sharma Kunapalli
sgk@cs.princeton.edu

Michael F. Cohen
mfc@cs.princeton.edu

Department of Computer Science
Princeton University

Abstract

This paper describes a methodology for using the matrix-vector multiply and scan conversion hardware present in many graphics workstations to rapidly approximate the optical flow in a scene. The optical flow is a 2-dimensional vector field describing the on-screen motion of each pixel. An application of the optical flow to MPEG compression is described which results in improved compression with minimal overhead.

CR Categories and Subject Descriptors: I.2.10 [Artificial Intelligence]: *Vision and Scene Understanding*; I.3.3 [Computer Graphics]: *Picture/Image Generation*; I.3.7 [Computer Graphics]: *Three-Dimensional Graphics and Realism*; I.4.2 [Image Processing]: *Compression (coding)*

Additional Key Words: MPEG; optical flow; motion prediction.

1 Introduction

The hardware based transformation, scan conversion and Gouraud shading interpolation in modern graphics workstations have been used for purposes beyond their original intent. Another use is described here, that of "rendering" an *optical flow* image[1], that is, an image in which the color of each pixel represents the motion occurring at that location in the image.

Optical flow has been used in computer vision [5, 8] and video compression [6]. It could also be useful for temporal anti-aliasing, providing information for the temporal sampling pattern [12] and for computing the *morph map* as discussed in [2]. After a short discussion of the optical flow computation itself, this paper outlines experiments using the motion information to enhance MPEG compression[2]. An application for this technology would be a networked *game server* in which players with machines containing low-cost MPEG decompression hardware are connected to a high speed graphics server over low bandwidth lines.

Computing the optical flow directly from an image sequence usually requires expensive search algorithms to match corresponding pixels. The approach taken here assumes the input has a polygonal representation and is being rendered on a frame buffer. The flow of vertices can then be calculated by taking the difference between past and present screen coordinates and encoding this information in the color channels. Gouraud shading then rapidly produces an approximate rendering of the optical flow of each pixel. Workstations equipped with texture mapping hardware can create a more accurate optical flow image.

A significant component of MPEG video compression is motion compensation, enabling one block of an image to point at a similar block in a previous (or future) frame. The optical flow field describes the location of each pixel in a previous frame, and thus provides a good starting point for finding a matching pixel block; using this technique reduces the search range necessary to achieve a desired level of compression.

[1] The term *motion field* is sometimes used to distinguish actual motion from apparent motion, e.g., a smooth shiny ball spinning in place in an otherwise static environment is moving but the motion won't be visible.

[2] CAD-based geometrical information has been used in other compression methods [3], however, with the increasing acceptance of the MPEG standard, there is interest in improving MPEG encoding itself.

2 Rendering the Optical Flow

When most graphics hardware renders a polygon with local shading, the colors are computed from lighting information at each vertex, and then bi-linearly interpolated across the polygon. Like-

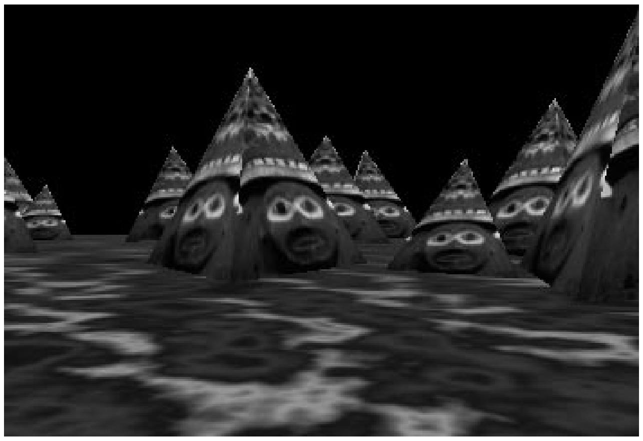

Figure 1: Frame from textured test-image sequence.

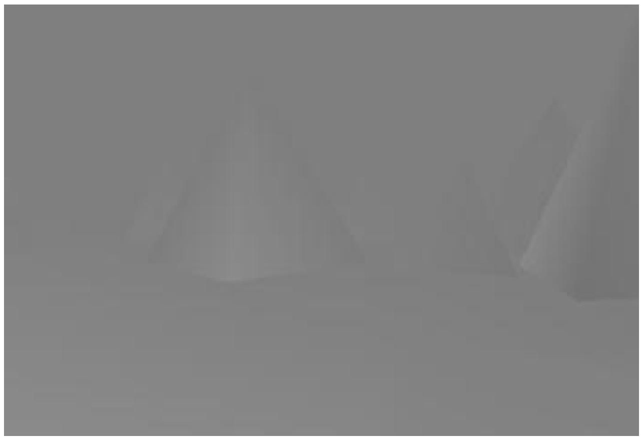

Figure 2: Optical flow image between two frames.

wise, for rendering the optical flow field, the optical flow vector is computed for each vertex, encoded as a color, and bi-linearly interpolated across the polygon using the same hardware. This requires knowing the transformation matrices which apply to each point in the previous frame as well as the current frame.

The Gouraud interpolation approach proceeds as follows. For each object, or group of polygons rendered with the same transformation matrix, the previous transformation matrices are saved so differences from them can be computed later. Let M and M' be the current and previous transformation matrices from world coordinates to screen coordinates. Ignoring clipping for the moment, a vertex v is transformed by multiplying by M, and dividing by w for the perspective transformation. The optical flow for a vertex going from transformation M to M' is

$$\Delta v_{\text{screen}} = X(v, M) - X(v, M')$$

where $X(v, M)$ represents the transformation operator. Unfortunately, $\Delta v_{\text{screen}} \neq X(v, M - M')$ since $M - M'$ may produce a zero perspective divide term w, causing division by zero. Using two 8-bit channels, $\Delta v_{\text{screen}} = (\Delta x, \Delta y)$ is coded into RGB space as:

$$(R, G, B) = (\Delta x + 127, \Delta y + 127, 127)$$

clipping large motions at 0 and 255 (i.e., a maximum of 127 pixels of motion per frame).

A Gouraud-shaded polygon is rendered with these colors at the vertices. Since the geometry and viewing transformations are the same as in normal rendering, polygons occupy the same pixels on the screen. The number of polygons remains the same, thus the optical flow is rendered in a single frame time (Figure 3).

Clipping causes a problem for polygons with some off-screen vertices. If a vertex is near or behind the viewer, the screen coordinates (and optical flow vectors) become extremely large or change sign. To ensure the accuracy of the optical flow vectors, the renderer may subdivide polygons near the viewer. For the experiment in this paper, the floor is subdivided into a regular grid and the pyramids are not subdivided.

However, Gouraud interpolation is only an approximation of the exact answer, for similar reasons that Gouraud shading based on scanline order only approximates object space bilinear interpolation (e.g., it is not rotationally invariant). To be more precise, each pixel would need to be transformed from the current frame to the previous frame, rather than just the vertices. The function to compute this transformation is a *projective mapping*[4].

Projective texture mapping normally utilizes an affine transformation from homogeneous coordinates to texture coordinates. For each polygon, the texture coordinates of the vertices are transformed by the texture matrix resulting in the indices into the texture map. These are then used during rasterization to pull out the texture value at each screen pixel. We can modify this process to encode the pixel's previous location as its texture value. The optical flow vector will then simply be the difference between this value and the new pixel location. This process can be accomplished using current texture mapping hardware [1] in real-time.

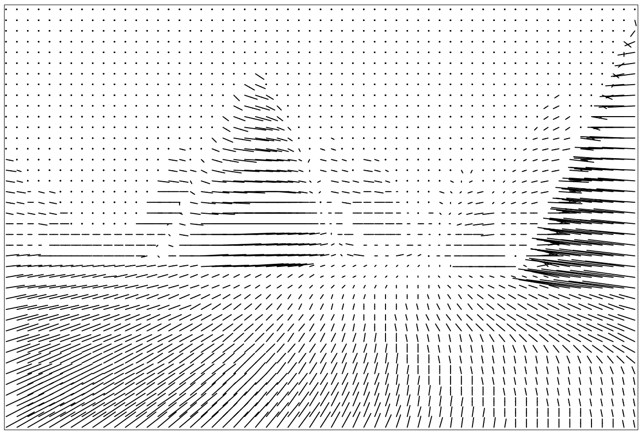

Figure 3: Vector-interpretation of an optical flow.

The algorithm runs as follows: the polygon's previous and current homogeneous coordinates are used to derive a texture matrix which maps between them. Thus, given the current coordinates, this matrix provides the homogeneous coordinates of the previous frame, and these are then used by the texture mapping hardware to find the "texture coordinates". Each polygon is rendered with an *identity texture*, which maps texture coordinates $(u, v) \rightarrow (u, v, 0)$, a color which directly encodes the texture coordinate[3]. The result is that each pixel's color encodes the screen location of that pixel in a previous frame. The optical flow field, $(\Delta x, \Delta y)$, is then found by subtracting off the pixel location from its encoded value. This too can be done in current hardware with an accumulation buffer:

$$\Delta v_{x,y} = \text{Screen}_{x,y} - (x, y)$$

where (x, y) is simply the identity texture itself. Computing the optical flow correctly with projective mapping is only practical with significant hardware support, but clipping and large polygons will be handled properly.

In general, accurately rendering the optical flow of a polygon is equivalent in complexity to rendering a texture-mapped polygon, and any techniques developed for texture map rendering may be applied to optical flow rendering.

3 Some MPEG Basics

This section focuses on the *motion estimation* component of MPEG. For introductory information on MPEG, see [6, 7, 10]. MPEG exploits the general similarity of adjacent picture frames by using *motion vectors* for each 16×16 pixel *macroblock* to point to another 16×16 block in a previous or future frame. The MPEG coder records the motion vector itself and the *differences* between previous and current pixels. Since many of the pixels will be largely the same, the difference will contain many zeros or small terms, requiring fewer bits to encode.

MPEG has three frame types:

- *I* (intra-coded or reference) frames are compressed without references to other frames. As a result they consume more bits, but have less error.
- *P* (or predicted) frames contain motion estimations from the preceding *I* frame, along with the difference between the two blocks. Because the difference is usually small, it compresses well.
- *B* (or bi-directional) frames have motion estimations from the nearest *P* or *I* frames in their past *and* future. The difference from the current block to the average of the past and future blocks is encoded. In addition, *B* frames are generally encoded with higher *quantization*, which increases compression at the cost of accuracy.

A typical MPEG sequence will follow a repeating pattern of these frame types such as *I*, *IPPP*, or *IBBPBB*. Figure 4 shows the dependencies in the *IBBPBB* pattern. An arrow from frame A to frame B implies that each macroblock in A should contain a motion vector describing a similar macroblock in B.

[3]Rendering images larger than 256×256 will require a framebuffer with more than 8 bits per channel to represent the motion vectors with sufficient precision.

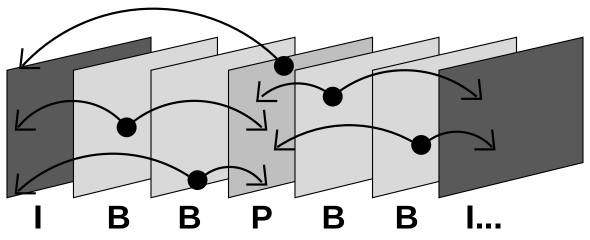

Figure 4: MPEG motion vector dependencies.

The literature on motion searching [9] discusses methods to reduce the number and/or cost of comparisons between blocks. Some well-known search techniques are:

- *Exhaustive* – select the best motion vector within a $2N \times 2N$ pixel square of pixels (best compression, but slow)
- *Logarithmic*[6] – first check the corners, sides, and center of a $2N \times 2N$ pixel square range, and recursively (until $N = 0$) repeat the procedure with the best result as the new center and N decremented (fast, but may overlook good solutions)
- *Subsample*[13] – rather than comparing whole macroblocks, only a fraction of the pixels in each macroblock are compared (fast, but may mistake a bad match as a good one)

Under normal circumstances, each algorithm centers its search range on the origin, assuming equal likelihood of motion in any direction.

4 Optical Flow and MPEG Compression

Given optical flow information, any of the search algorithms from the previous section can be improved by centering their search at the optical flow's *predicted motion*. This technique should improve *any* block matching scheme with limited search range.

Modifications to the Berkeley MPEG encoder's [11] exhaustive search algorithm were made to measure the effects of shifting the center of the search range. The motion prediction for a macroblock is taken as the *mode* of the associated 16×16 array of optical flow data. The *mode* rather than the *mean* is used since, if a macroblock contains two or more polygons, the polygon with the largest area is likely to provide a better motion estimate than an average of the two.

Test data from a 30-frame sequence of motion through an environment was used (Figure 1). Experiments included flat-shaded versus textured polygons, and static versus dynamic geometry. The optical flow buffers were rendered with the Gouraud-interpolation method. Camera motion was the same in all cases. Compression was measured as a function of search range[4].

The predicted motion produced modest improvement with flat-shaded input, however, with textures the motion prediction compressed significantly better than the other methods (Figure 6). As the search range increases, the two methods converge as expected (i.e., if searching the whole image, any method could find the best match).

5 Conclusion

A fast means to compute approximate optical flow images of polygonal data with minimal changes to the standard graphics pipeline has been demonstrated. One application of the results, MPEG compression, has been shown to benefit from this additional information, particularly for textured scenes.

[4] More results are available through the World Wide Web (e.g.: *xmosaic*). `http://www.cs.princeton.edu/grad/dwallach/sg94`

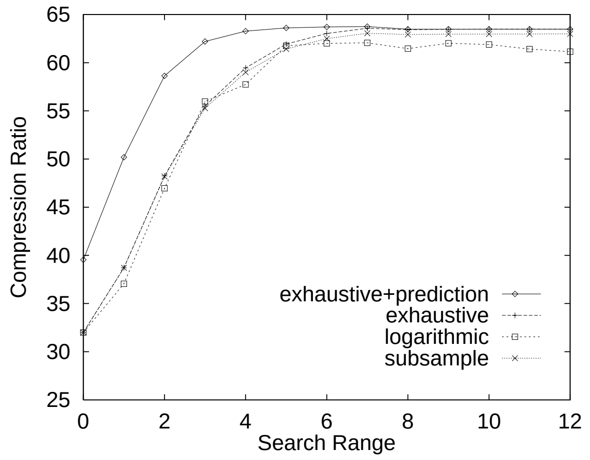

Figure 5: Compression vs. search range for texture-mapped polygons with static scenery using an IPPP pattern.

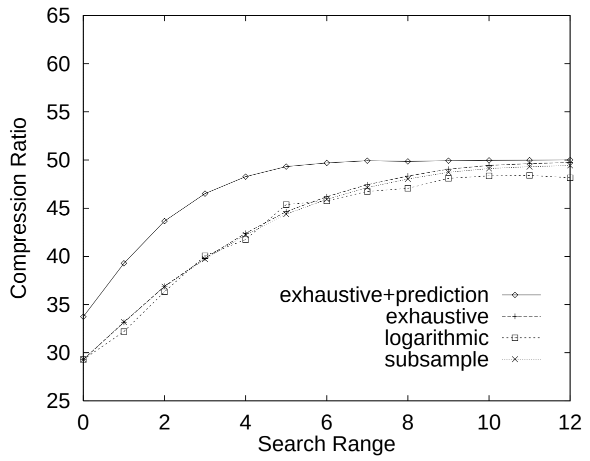

Figure 6: Textured polygons, dynamic scenery, IPPP pattern.

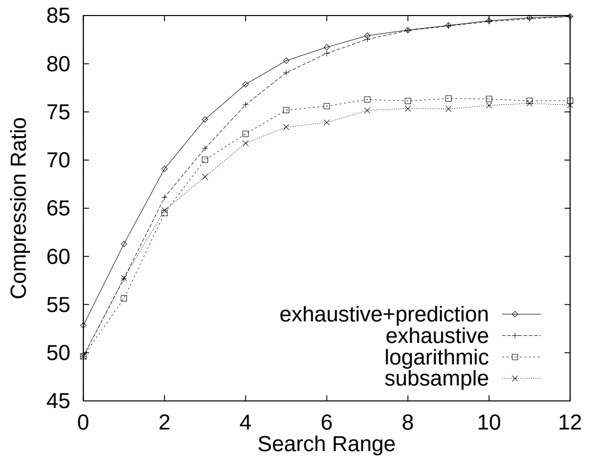

Figure 7: Flat-shaded polygons, dynamic scenery, IPPP pattern.

The use of texture mapping hardware to handle the projective mapping of motion coordinates to colors needs to be investigated to fully take advantage of current workstation capabilities.

Acknowledgements

This work has relied to great extent on the publically available MPEG encoder from U. C. Berkeley. Pat Hanrahan helped work through the equivalence of the mapping to the texture mapping problem. Philip Greenspun provided the Alaskan totem pole picture. The authors would like to thank the reviewers for their excellent suggestions leading to the current paper.

References

[1] Akeley, K. RealityEngine Graphics. In *Computer Graphics* (Aug. 1993), pp. 109–116.

[2] Chen, S. E., and Williams, L. View Interpolation for Image Synthesis. In *Computer Graphics* (Aug. 1993), pp. 279–288.

[3] Guenter, B. K., Yun, H. C., and Mersereau, R. M. Motion Compensated Compression of Computer Animation Frames. In *Computer Graphics* (Aug. 1993), pp. 297–304.

[4] Heckbert, P. S. Fundamentals of Texture Mapping and Image Warping. Master's thesis, University of California, Berkeley, June 1989.

[5] Horn, B. K. P. *Robot Vision*. MIT Press, 1986.

[6] Coded Representation of Picture, Audio and Multimedia/Hypermedia Information. Committee Draft of Standard ISO/IEC 11172, Dec. 1991.

[7] Legall, D. MPEG – A Video Compression Standard for Multimedia Applications. *CACM 34*, 4 (Apr. 1991), 46–58.

[8] Nagel, H. On the Estimation of Optical Flow: Relations Between Different Approaches And Some New Results. In *Artificial Intelligence* (1987), vol. 33, pp. 299–324.

[9] Orchard, M. A Comparison of Techniques for Estimating Block Motion in Image Sequence Coding. In *Proceedings SPIE, Visual Comm. and Image Proc. I* (1989), vol. 1199, pp. 248–258.

[10] Patel, K., Smith, B. C., and Rowe, L. A. Performance of a Software MPEG Video Decoder. In *Multimedia '93 Proceedings* (Aug. 1993), ACM, Addison-Wesley, pp. 75–82.

[11] Rowe, L. A., Gong, K., Patel, K., and Wallach, D. MPEG-1 Video Software Encoder. Anonymous ftp `mm-ftp.cs.berkeley.edu:/pub/multimedia/mpeg/mpeg_encode-1.3.tar.Z`, Mar. 1994.

[12] Shinya, M. Spatial Anti-aliasing for Animation Sequences with Spatio-temporal Filtering. In *Computer Graphics* (Aug. 1993), pp. 289–296.

[13] Zaccarin, A., and Liu, B. Fast Algorithms for Block Motion Estimation. In *Proceedings ICASSP'92* (Mar. 1992), pp. III–449 – III–452.