Deep Compression for Streaming Texture Intensive Animations

Daniel Cohen-Or [*] Tel Aviv University
Yair Mann [‡] Webglide Ltd.
Shachar Fleishman [*] Tel Aviv University

Abstract

This paper presents a streaming technique for synthetic texture intensive 3D animation sequences. There is a short latency time while downloading the animation, until an initial fraction of the compressed data is read by the client. As the animation is played, the remainder of the data is streamed online seamlessly to the client. The technique exploits frame-to-frame coherence for transmitting geometric and texture streams. Instead of using the original textures of the model, the texture stream consists of view-dependent textures which are generated by rendering offline nearby views. These textures have a strong temporal coherency and can thus be well compressed. As a consequence, the bandwidth of the stream of the view-dependent textures is narrow enough to be transmitted together with the geometry stream over a low bandwidth network. These two streams maintain a small online cache of geometry and view-dependent textures from which the client renders the walkthrough sequence in real-time. The overall data transmitted over the network is an order of magnitude smaller than an MPEG post-rendered sequence with an equivalent image quality.

Keywords: compression, MPEG, streaming, virtual environment, image-based rendering

1 Introduction

With the increasing popularity of network-based applications, compression of synthetic animation image sequences for efficient transmission is more important than ever. For long sequences even if the overall compression ratio is high, the latency time of downloading the compressed file might be prohibitive. A better network-based compression scheme is to partition the compressed sequence into two parts. The first part, or header, is small enough to be downloaded within an acceptable initialization time, while the second part is transmitted as a *stream*. The compressed data is broken up into a stream of data which is processed along the network pipeline, that is, the compressed data is transmitted from one end, and received, decoded and displayed at the other end. Streaming necessarily requires that all the pipeline stages operate in real-time. Clearly, the network bandwidth is the most constrained resource along the pipeline and thus the main challenge is to reduce the stream bandwidth enough to accommodate the network bandwidth constraint.

Standard video compression techniques that take advantage of frame-to-frame coherency, are insufficient for streaming. Given a moderate frame resolution of 256×256, an average MPEG frame is typically about 2-6K bytes. Assuming a network with a sustained transfer rate of 2K bytes per second, a reasonable quality of a few frames per second cannot be achieved. A significant improvement in compression ratio is still necessary for streaming video in real-time.

Synthetic animations have higher potential to be compressed than general video since more information is readily available for the compression algorithms [2, 12, 22]. Assuming, for example, that the geometric model and the animation script are relatively small, one can consider them as the compressed sequence, transmit them and decode them by simply rendering the animation sequence. By simultaneously transmitting the geometry and the script streams on demand, one can display very long animations over the network (see [19]). However, if the model consists of a large number of geometric primitives and textures, simple streaming is not enough. The texture stream is especially problematic since the texture data can be significantly larger than the geometric data. For some applications, replicated (tiled) textures can be used to avoid the burden of large textures. However, realistic textures typically require a large space. Moreover, detailed and complex environments can effectively be represented by a relatively small number of textured polygons [3, 13, 17, 20, 6, 21, 11]. For such *texture intensive* models the geometry and the animation script streams are relatively small, while the texture stream is prohibitively expensive. Indeed, the performance of current VRML browsers is quite limited when handling texture intensive environments [8].

In this paper we show that instead of streaming the textures, it is possible to stream *view-dependent textures* which are created by rendering nearby views. These textures have a strong temporal coherency and can thus be well compressed. As a consequence, the bandwidth of the stream of the view-dependent textures is narrow enough to be transmitted together with the geometry and the script streams over a low bandwidth network. Our results show that a walkthrough in a virtual environment consisting of tens of megabytes of textures can be streamed from a server to a client over a network with a sustained transfer rate of 2K bytes per second. In terms of compression ratios, our technique is an order of magnitude better than an MPEG sequence with an equivalent image quality.

2 Background

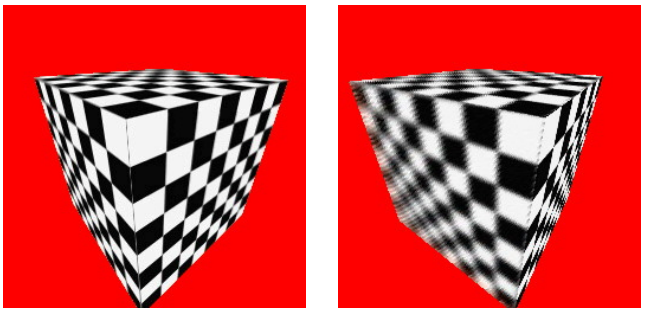
Standard video compression techniques are based on image-based motion estimation [9]. An MPEG video sequence consists of intra frames (I), predictive frames (P) and interpolated frames (B). The I frames are coded independently of any other frames in the sequence. The P and B frames are coded using motion estimation and interpolations, and thus, they are substantially smaller than the I frames. The compression ratio is directly dependent on the success of the motion estimation for coding the P frames. Using image-based techniques the optical flow field between successive frames can be approximated. It defines the motion vectors, or the correspondence between the pixels of successive frames. To reduce the overhead of encoding the motion information, common techniques compute a motion vector for a block of pixels rather than for individual pixels. For a synthetic scene the available model can assist in computing the optical flow faster or more accurately [2, 22]. These model-based motion estimation techniques improve the compression ratio, but the overhead of the block-based motion approximation is still too high for streaming acceptable image quality.

A different approach was introduced by Levoy [12]. Instead of compressing post-rendered images, it is possible to render the animation on-the-fly at both ends of the communication. The rendering task is partitioned between the server (sender) and client (receiver). Assuming the server is a high-end graphics workstation, it

[*] Computer Science Department, Tel-Aviv University 69978, Israel. E-mail: {daniel, shacharf}@math.tau.ac.il
[‡] Webglide Ltd. E-mail yair@webglide.com

can render high and low quality images, compute the residual error between them, and transmit the residual image compressed. The client needs to render only the low quality images and to add the transmitted residual images to restore the full quality images. It was shown that the overall compression ratio is better than conventional techniques.

If this technique is to be adapted for streaming of the residual images, it would require pre-computing the residual images and downloading the entire model and animation sequence before streaming can take place. It would be possible to transmit only key-frame images and blend (or extrapolate) the in-between frames [4, 14, 15]. However, it is not clear how to treat texture intensive animations, since downloading the textures is still necessary to keep the residual images small enough for streaming. Other related techniques use *imposters* [17, 20] and *sprites* [18, 11] as image-based primitives to render nearby views. When the quality of a primitive drops below some threshold, it is recomputed without exploiting its temporal coherence.



(a) source view (b) mapped view

Figure 1: (a) A source image of a textured cube. (b) A view of the same cube obtained by backmapping into the source image. Note that the texture on the leftmost face is severely blurred, while the other two faces do not exhibit significant artifacts.

3 View-dependent Texture Streaming

Assume an environment consisting of a polygonal geometric model, textures, and a number of light sources. Furthermore, assume that the environment is texture-intensive, that is, the size of the environment database is dominated by the textures, while the geometry-space is significantly smaller than the texture-space. Nevertheless, the size of the environment is too large to be downloaded from the server to the client in an acceptable time.

Streaming the environment requires the server to transmit the animation script and, according to the camera viewing parameters, to transmit the visible parts of the model. However, the size of the textures necessary for a single view is too large to be streamed in real-time. Instead of using these original textures, we show that it is better to use nearby views as textures. These views can be regarded as view-dependent textures which are effectively valid only for texturing the geometry viewed from nearby viewing directions.

Given the current camera position, the client generates a new frame based on the data streamed so far, which includes at least the visible polygons from the current camera position and a number of nearby views. The new frame is a perspective view generated by rendering the geometry where the nearby perspective views serve as texture maps. Since the correspondence between two perspective views is a projective map, the model is rendered by applying a projective mapping rather than a perspective mapping. This projective map can be expressed by a linear transformation in homogeneous space, and can be implemented by rational linear interpolation which requires divisions at each pixel [10, 16].

Nearby views are advantageous as textures because (i) they have an "almost" one-to-one correspondence with the current view and therefore, a bi-linear filter, is sufficient to produce quality images, and (ii) they are post-rendered and may include various global illumination effects. Figure 1 shows a simple example of a textured cube. In these views there are three visible polygonal faces. The left image was rendered with a standard perspective texture mapping, while the right image was rendered by applying a backward projective texture mapping from the left image. The projection of each of the cube faces on the left image serves as a view-dependent texture for its correspondence projection on the right image. The size and shape of the source texture is close to its target. This reduces many aliasing problems and hence simplifies the filtering. In this example, for expository reasons, the two views are not "nearby", otherwise they would appear almost indistinguishable. Later we will quantify the meaning of "nearby". Since the view-dependent textures are generated off-line, they can be involved and can account for many global illumination effects, including shadows cast from remote objects. This simplifies the generation of a new frame, since the rendering is independent of the complexity of the illumination model.

View-dependent textures, like view-independent textures are discrete, therefore, when resampled, artifacts are introduced. If the target area is larger than the corresponding source area the result appears blurry, as can be seen on the leftmost face of the cube in Figure 1b. On the other hand, if the target area is smaller than the source area (the rightmost face), by appropriately filtering the source samples, the target image has no noticeable artifacts. Thus, an appropriate view-dependent texture should be (i) created as close as possible to the new view and (ii) created from a direction in which the area is expanded relative to the target. In Section 4 we elaborate on the factor which controls the quality of the view-dependent textures.

One should also consider the visibility, namely that at least part of a polygon can be hidden at the nearby view, while being visible at the current view. This is a well-known problem in image-based rendering [4, 5]. However, in a streaming application where the compressed sequence is precomputed, visibility gaps are known a priori and for each gap there is at least one nearby view that can close that gap. It should be emphasized that the nearby views are not restricted to be from the "past", but can be located near a "future" location of the camera.

As the animation proceeds, the quality and validity of the view-dependent textures decrease as the camera viewing parameters deviate. Thus, new closer view-dependent textures need to be streamed to guarantee that each polygon can be mapped to an appropriate texture. The stream of new view-dependent textures is temporally coherent and thus instead of transmitting the compressed textures, it is possible to exploit this texture-to-texture coherence to get a better compression. The old texture is warped towards the new view and the residual error between the new view and the warped image is computed. The residual image can then be better compressed than the raw texture.

A view-dependent texture is not necessarily a full size nearby view, but may include only the view of part of the environment. Based on a per-polygon amortization factor (see Section 4), a subset of the view-dependent textures, defined by the polygon mesh, is updated. This further reduces the size of texture stream to be optimal with respect to the amortization factor. Thus, a key point is to use a good texture quality estimate.

4 The Texture Quality Factor

Given a view and a set of view-dependent textures, it is necessary to select the best quality texture for each visible polygon. As can be seen in Figure 1 for some faces of the cube (in (b)) the view-dependent texture (in (a)) is more appropriate than others. The quality of the source texture is related to the area covered by the projection of each polygon, or cube face in Figure 1, in the two views. Thus, a per polygon local *scaling factor* is required to estimate the areas in the source texture that locally shrink or expand when mapped onto the target image. When the scaling factor is ≤ 1 we can consider the source image appropriate for generating the target image. As the factor increases above 1, more and more blur appears in the target image, and we should consider the source texture less and less appropriate. This per polygon texture quality factor is used to select the best source texture out of the available set of view-dependent textures. If the best source is above some predetermined threshold, a new texture is required. However, a successful streaming of the textures guarantees that there is always one available texture whose quality factor is satisfactory.

Figure 2 illustrates the per-polygon source selection process. Two examples of nearby views are shown in (a) and (b). The scale factor for the in-between images (f) is visualized using the gray level images (c) and (d), where white is used for low scale factor and black is used for high scale factor. The (c) and (d) columns visualize the scale factor when mapping the textures from (a) and (b) to (f). For each polygon, we select the texture from the image that has the lowest scale factor, this is visualized in (e), where the red levels are the scale factors from (a) and the blue levels are the scale factor from (b).

We are now left with the problem of estimating the maximal value of the scaling factor in a particular polygon in the source image, for a given target image (see also [11]). Note that the factor is defined independently of the visibility of the polygon. We first discuss the case where the polygon is mapped from the source to the target by a linear transformation, and then discuss the more general non-linear case.

Let A be a square matrix corresponding to some linear transformation. The scaling factor of a linear transformation is the 2-norm of the matrix A, i.e. the maximum 2-norm of Av over all unit vectors v.

$$\max_{||v||_2=1} \|Av\|_2$$

It can be shown [7] that the 2-norm of A equals the square root of λ_{max}, the largest eigenvalue of $A^{\mathrm{T}}A$. In the case of two-dimensional linear transformations, where A is a 2 by 2 matrix, we can write down a closed-form expression for λ_{max}. Let a_{ij} denote the elements of A and e_{ij} the elements of $A^{\mathrm{T}}A$:

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \qquad A^{\mathrm{T}}A = \begin{pmatrix} e_{11} & e_{12} \\ e_{21} & e_{22} \end{pmatrix} \tag{1}$$

The eigenvalues of the matrix $A^{\mathrm{T}}A$ are the roots of the polynomial $\det(A^{\mathrm{T}}A - \lambda I)$, where I is the identity matrix. In the two-dimensional case, λ_{max} is the largest root of the quadratic equation

$$(e_{11} - \lambda)(e_{22} - \lambda) - e_{12}e_{21} = 0. \tag{2}$$

Thus,

$$\lambda_{max} = \frac{e_{11} + e_{22} + \sqrt{(e_{11} + e_{22})^2 - 4(e_{11}e_{22} - e_{12}e_{21})}}{2}. \tag{3}$$

Expressing the elements e_{ij} in terms of the elements a_{ij} yields

$$A^{\mathrm{T}}A = \begin{pmatrix} a_{11}^2 + a_{21}^2 & a_{11}a_{12} + a_{21}a_{22} \\ a_{11}a_{12} + a_{21}a_{22} & a_{12}^2 + a_{22}^2 \end{pmatrix}, \tag{4}$$

and finally, defining $S = \frac{1}{2}(a_{11}^2 + a_{12}^2 + a_{21}^2 + a_{22}^2)$, we get

$$\lambda_{max} = S + \sqrt{S^2 - (a_{11}a_{22} - a_{12}a_{21})^2} \tag{5}$$

Dealing with non-linear transformations, such as projective transformations, requires measuring the scaling factor locally at a specific point in the image by using the partial derivatives of the transformation at the given point. The partial derivatives are used as the coefficients of a linear transformation.

Let us denote by x_0, y_0, and x_1, y_1 the source and target image coordinates of a point, respectively, and by x, y, z the three-dimensional location of that point in target camera coordinates. We have:

$$(x, y, z)^{\mathrm{T}} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} (x_0, y_0, 1)^{\mathrm{T}} \tag{6}$$

$$(x_1, y_1)^{\mathrm{T}} = \frac{1}{z}(x, y)^{\mathrm{T}} \tag{7}$$

or explicitly,

$$x_1 = \frac{ax_0 + by_0 + c}{gx_0 + hy_0 + i} \qquad y_1 = \frac{dx_0 + ey_0 + f}{gx_0 + hy_0 + i} \tag{8}$$

The partial derivatives of the above mapping at (x_1, y_1) define its gradient, which is a linear transformation:

$$A = \begin{pmatrix} \partial x_1/\partial x_0 & \partial x_1/\partial y_0 \\ \partial y_1/\partial x_0 & \partial y_1/\partial y_0 \end{pmatrix} = \frac{1}{z^2} \begin{pmatrix} za - x_1 g & zb - x_1 h \\ zd - y_1 g & ze - y_1 h \end{pmatrix} \tag{9}$$

In cases where the field of view is small, and there is only a little rotation of the plane relative to the source and target views, the following approximation can be used. The transformation of the plane points from source to target image can be approximated by:

$$\begin{aligned} x_1 &= a + bx_0 + cy_0 + gx_0^2 + hx_0y_0 \\ y_1 &= d + ex_0 + fy_0 + gx_0y_0 + hy_0^2 \end{aligned} \tag{10}$$

This is called a pseudo 2D projective transformation and is further explained in [1]. In this case we get:

$$A = \begin{pmatrix} b + 2gx_0 + hy_0 & c + hx_0 \\ e + gy_0 & f + gx_0 + 2hy_0 \end{pmatrix} \tag{11}$$

To estimate the maximal scaling factor, the gradient can be computed at the three vertices of a triangle. In cases where the triangle is small enough, even one sample (at the center of the triangle) can yield a good approximation. Our experiments have shown that a scale factor in the range of $[1.3, 1.4]$, yields high quality images while maintaining high compression ratio.

5 Geometry Streaming

We have discussed the strategy for selecting the best view-dependent texture from those available. The system guarantees that at least one appropriate view-dependent texture has already been streamed to the client. If for some reason an appropriate texture is not available on time, some blur is likely to be noticed. However, the error is not critical. On the other hand, a missing polygon might cause a hole in the scene and the visual effect would be unacceptable. Thus, the geometry stream gets a higher priority so as to avoid any geometric miss.

Detecting and streaming the visible polygons from a given neighborhood is not an easy problem. Many architectural models

are inherently partitioned into cells and portals by which the cell-to-cell visibility can be precomputed. However, since the geometry stream is computed offline, the visible polygons can be detected by rendering the entire sequence. The set of polygons visible in each frame can be detected by rendering each polygon with ID index color and then scanning the frame buffer collecting all the ID's of the visible polygons. The definition of a polygon that is first visible is transmitted together with its ID. During the online rendering the client maintains a list of all the polygons which have been visible so far, and a cache of the "hot" ID's, i.e., the polygons which are either visible in the current frame or in a nearby frame. The temporal sequence of the hot ID's is precomputed and is a part of the geometric stream. The cache of hot polygons serves as a visibility culling mechanism, since the client needs to render only a small conservative superset of the visible polygons, which is substantially smaller than the entire model.

Table 1: A comparison of the streaming technique (VDS) with an MPEG encoding. The frame resolution is 240×180.

animation (# frames)	size (header)	RMS (0-255)	bits/pix	bits/sec
Arts (435)				
VDS	80K (23K)	18.6	0.034	29511
MPEG	1600K	19.2	1.61	619168
MPEG	200K	31	0.085	73766
Gallery (1226)				
VDS	111K (30K)	19.15	0.0168	14563
MPEG	1830K	20.03	0.277	239436
MPEG	333K	27.41	0.05	43523
Office (1101)				
VDS	177K (72K)	22.47	0.03	25722
MPEG	2000K	22.60	0.336	290644
MPEG	337K	34.71	0.056	48937

6 Results and Conclusions

In our implementation the precomputed sequence of view-dependent textures is streamed to the client asynchronously. The stream updates the set of view-dependent textures from which the client renders the animation. The texture update frequency is defined to guarantee some reasonable quality. The choice of frequency directly effects the compression ratio. Reasonable quality is a subjective jusgement, however we will analyze it quantitatively. We use the *root mean square* (RMS) as an objective measurement to compare MPEG compression and our compression technique (denoted in the following by VDS (View Dependent texture Stream)) with the uncompressed images.

Figure 3 shows five samples of the comparison taken from three sequences that were encoded both in MPEG and VDS. The first two examples, "Arts" and "Gallery", are from two virtual museum walkthroughs, and the third example "Office" is of an animation. The quantitative results are shown in Table 1. Each animation was compressed by MPEG and VDS; we see that when they have about the same RMS, the size of the MPEG is about 10-20 times larger than the VDS. When MPEG is forced to compress the sequence down to a smaller size, it is still 2-3 times larger than the VDS, and its quality is significantly worse. This is evident from the corresponding RMS values in the table and visually noticeable in Figure 3. The VDS is not free of artifacts; some are apparent in the images in Figure 3. Most of them are due to imperfect implementation of the rendering algorithm, rather than being intrinsic to the algorithm. Note that there are slight popping effects when a texture update changes the resolution, which could have been avoided by using a blending technique.

The texture intensive models were generated with 3D Studio Max. The virtual museum "Arts", consists of 3471 polygons and 19.6 Megabytes of texture, and the "Office" animation consists of 5309 polygons and 16.9M Megabytes of 3D Studio Max's textures, that were not optimized for size. Part of the compressed data is the header, which includes the initial data necessary to start the local rendering, that is, the geometry, the animation script, and the first nearby view in JPEG format. Thus, the cost effectiveness of streaming is higher for longer animations. The size of the header appears in parenthesis near the size values in Table 1.

We implemented both the client and the server on a Pentium machine, which was integrated in a browser with ActiveX. The client and the server were connected by a dial-up connection of 14400 bits per second. The download time of the initial setup data, namely the header, is equivalent to the download time of a JPEG image. This means a few seconds on a network with a sustained transfer rate of less than 2K bytes per second. We attached hot-spots (hyperlinks) to the 3D objects visible in the animation, by which the user chooses either high resolution images or precomputed sequences which are streamed online according to his selection.

The offline process that generates the VDS sequence takes a few minutes for animations of several hundred frames as in the above examples. Currently, the client's local rendering is the slowest part of the real-time loop. It should be emphasized that the client uses no graphics hardware to accelerate the rendering which requires a perspective texture mapping. We believe that optimizing the client rendering mechanism will permit streaming sequences of higher resolutions. We are planning to extend our system by allowing some degree of freedom in the navigation, for example, by supporting branching from one movie sequence to others. Currently, we are working on applying computer-vision techniques to reconstruct the geometry of real-world movies and compress them using our geometry-based compression. We believe that the importance of geometry-based compression is likely to play an increasing role in 3D network-based applications.

References

[1] G. Adiv, Determining three-dimensional motion and structure from optical flow generated by several moving objects. in *IEEE Transactions on PAMI*, 7(4):384-401, July 1985.

[2] M. Agrawala, A. Beers and N. Chaddha. Model-based motion estimation for synthetic animations. *Proc. ACM Multimedia '95*.

[3] D. G. Aliaga. Visualization of complex models using dynamic texture-based simplification. *Proceedings of Visualization 96*, 101–106, October 1996.

[4] S.E. Chen and L. Williams. View interpolation for image synthesis. *Computer Graphics (SIGGRAPH '93 Proceedings)*, 279–288, August 1993.

[5] L. Darsa and B. Costa and A. Varshney. Navigating static environments using image-space simplification and morphing. *1997 Symposium on Interactive 3D Graphics*, Providence, Rhode Island; April 28–30, 1997.

[6] P.E. Debevec, C.J. Taylor and J. Malik. Modeling and Rendering Architecture from Photographs: A Hybrid Geometry- and Image-Based Approach. *Computer Graphics (SIGGRAPH '96 Proceedings)*, 11–20, August 1996.

[7] G.H. Golub, C.F. Van Loan. Matrix Computations. Second edition, pp. 56–60.

[8] J. Hardenberg, G. Bell and M. Pesce. VRML: Using 3D to surf the web. SIGGRAPH'95 course, No. 12 (1995).

[9] D. LeGall. MPEG: A video compression standard for multimedia applications. in *Communications of the ACM*, Vol 34(4), April 1991, pp. 46–58.

[10] P.S. Heckbert and H.P. Moreton. Interpolation for polygon texture mapping and shading. *State of the Art in Computer Graphics: Visualization and Modeling.* pp. 101–111, 1991.

[11] Jed Lengyel and John Snyder. Rendering with Coherent Layers, *Computer Graphics (SIGGRAPH 97 Proceedings)*, 233-242, August 1997.

[12] M. Levoy. Polygon-assisted JPEG and MPEG compression of synthetic images. *Computer Graphics (SIGGRAPH '95 Proceeding)*, 21–28, August 1995.

[13] P.W.C. Maciel and P. Shirley. Visual navigation of large environments using textured clusters. *1995 Symposium on Interactive 3D Graphic*, 95–102, April 1995.

[14] Y. Mann and D. Cohen-Or. Selective pixel transmission for navigating in remote virtual environments. in *Computer Graphics Forum (Proceedings of Eurographics'97)*, Volume 16(3), 201–206, September 1997.

[15] W.R. Mark, L. McMillan, G. Bishop. Post-rendering 3D warping. *Proceedings of 1997 Symposium on Interactive 3D Graphic*, Providence, Rhode Island; April 28–30, 1997.

[16] M. Segal, C Korobkin, R. van Widenfelt, J. Foran and P. Haeberli. Fast shadows and lighting effects using texture mapping. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 249–252, 1992.

[17] G. Schaufler and W. Sturzlinger. A three dimensional image cache for virtual reality. in *Computer Graphics Forum (Proceedings of Eurographics'96)*, Volume 15(3), 227–235, 1996

[18] G. Schaufler. Per-Object Image Warping with Layered Impostors. in Proceedings of the 9th Eurographics Workshop on Rendering '98, Vienna, 145–156, June 1998.

[19] D. Schmalstieg and M. Gervautz. Demand-driven geometry transmission for distributed virtual environment. *Eurographics '96, Computer Graphics Forum*, Volume 15(3), 421-432, 1996.

[20] J. Shade, D. Lischinski, D.H. Salesin, J. Snyder and T. DeRose. Hierarchical image caching for accelerated walkthroughs of complex environments. *Computer Graphics (SIGGRAPH '96 Proceedings)*,

[21] F. Sillion, G. Drettakis, and B. Bodelet. Efficient impostor manipulation for real-time visualization of urban scenery, in *Computer Graphics Forum (Proceedings of Eurographics'97)*, Volume 16(3), 207–218, September 1997.

[22] D.S. Wallach, S. Kunapalli and M.F. Cohen. Accelerated MPEG compression of dynamic polygonal scenes. *Computer Graphics (SIGGRAPH '94 Proceedings)*, 193–197, July 1994.

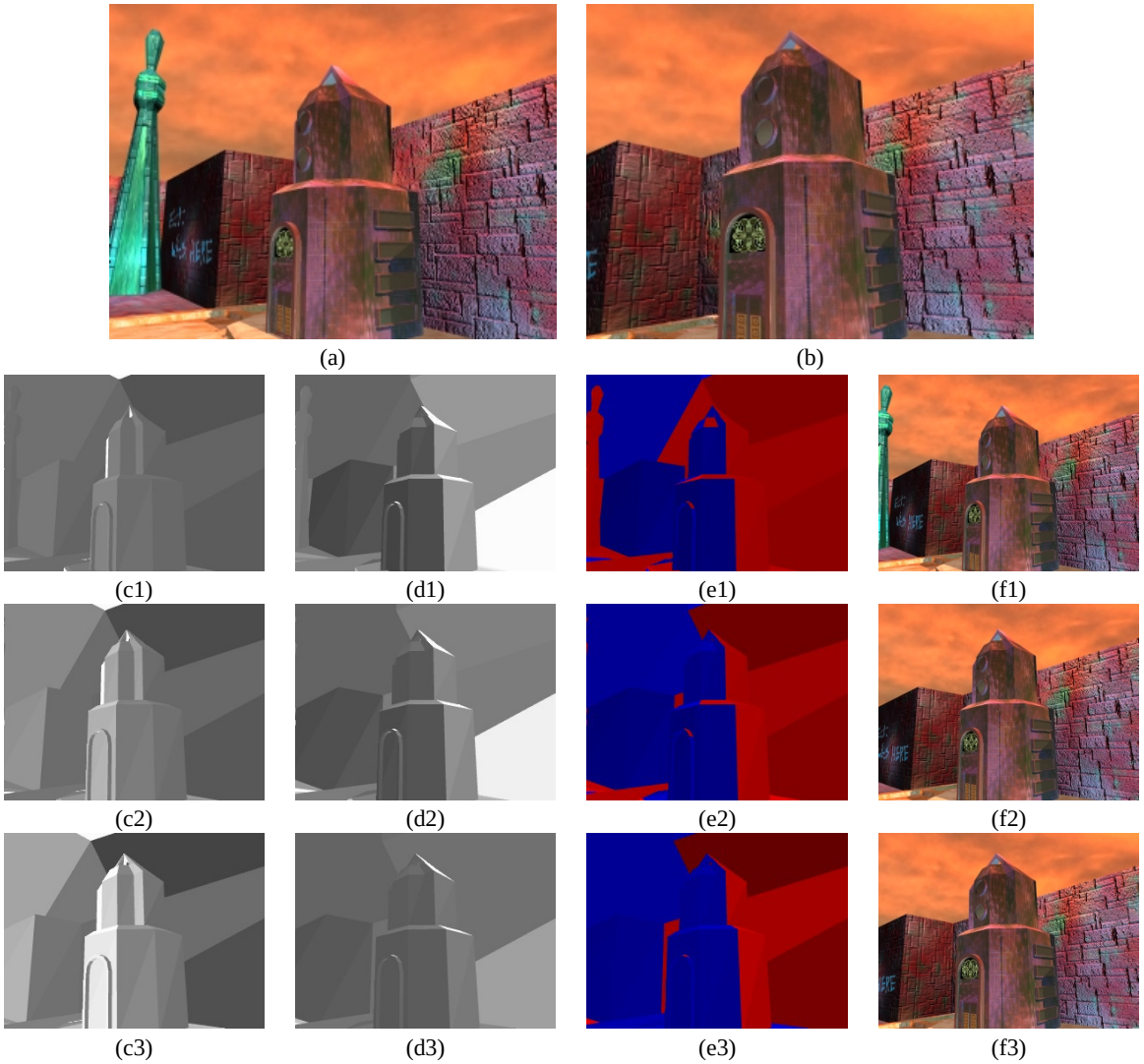


Figure 2: The scale factor. (a) and (b) are the two nearby views, (c) and (d) the per polygon scale factor in gray level from (a) and (b), respectively, (e) the red level and blue level polygons are mapped from (a) and (b), respectively, (f) three inbetween images.

Figure 3: A comparison between VDS and MPEG. The images in the left column are MPEG frames and in the right column are VDS frames. The size of MPEG sequence is about three times larger than the VDS sequence.