Progressive Hiding of a 3D Object into its Texture Image

Emmanuel Garcia, Jean-Luc Dugelay, Vanessa Lopez Eslava
Institut Eurécom
2229 route des crêtes, BP 193
06904 Sophia Antipolis, France

***Abstract*— This paper presents an original data-hiding application where the payload is intimately related to the host data. On the one hand we want to preserve this relationship so that each part of the payload is hidden in the related part of the host. On the other hand we want to ensure that a degradation of a part of the host data implies a proportional degradation of the related part of the hidden data.**

I. Introduction

Three-dimensional video objects belong to the so-called category of new objects in watermarking and/or data-hiding [4], [5].

In this paper we propose an algorithm to hide the geometry of a 3D object into its texture image. The result is that only the image is needed to transmit and build back the 3D object (see Fig. 1).

Data-hiding algorithms embed one document into another one [1], [2], [3], but usually there is no natural relation between them [7]. On the contrary our algorithm does not merely hide 3D information in an image without relating the one to the other, as if the 3D information were only an anonymous payload: since pixels of the texture are in a one-to-one correspondance with points of the surface of the object, there is a natural relationship between geometry and texture, as if the geometry were a 4th band for the three color bands of the texture, and we want to hide and interlace this 4th band into the three others.

By preserving the spatial relationship between payload (=geometry) and cover (=texture), if part of the texture is destroyed, the corresponding geometry is also destroyed, but the geometry corresponding to the untouched part of the texture remains recoverable and intact.

In addition to this property we want to perform the data-hiding in such a way that a degradation of the cover implies a proportional degradation of the payload. Overall the success of our scheme depends on the achieved degree of spatial synchronization and degradation correlation between the payload and the cover.

Note that some applications have one of these two properties, e.g. spatial (or rather temporal) coherence, as in [3], or correlated degradation, as in [6], but not both properties.

II. Representation of 3D data

In order to ensure the spatial relationship between the payload and the cover, we represent a 3D object as a cylindrical texture image and an associated cylindrical depth map.

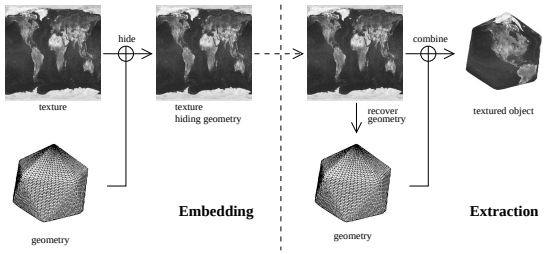


Fig. 1. Overview of our data-hiding scheme.

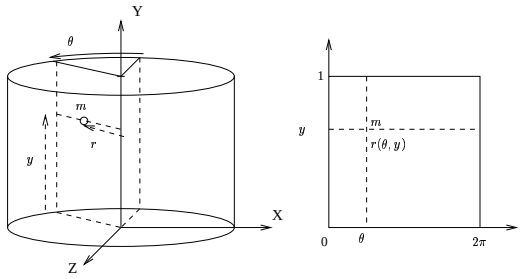
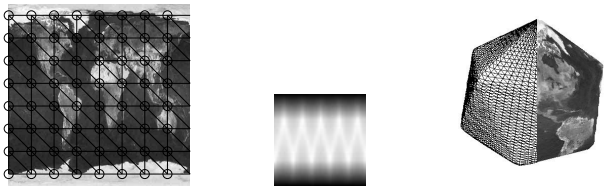


Fig. 2. From cartesian coordinates (X, Y, Z) to cylindrical coordinates (θ, y, r) to bidimensionnal depth map $r(\theta, y)$.

Furthermore, the resolution of the depth map is 8 times lower than the resolution of the texture image in both directions. Therefore each pixel of the depth map corresponds to a 8×8 block of the texture. The texture image is originally a color image, but the host information for data-hiding is the luminance of the image, computed from the Red, Green and Blue color components using the simplified formula $Y = 0.3R + 0.6G + 0.1B$.

Hiding each depth value into the corresponding texture block would achieve the desired property of preserving the spatial correlation between payload and cover (Fig. 4), but not the property of correlated degradation. Therefore, prior to any processing, we perform a three-level wavelet decomposition of both depth and texture. In this way it will be possible to hide low depth frequencies into low texture frequencies and high depth frequencies into high texture frequencies. Therefore the most significant depth coefficients (low frequencies) would be hidden in the most robust texture coefficients (low frequencies). When the texture is progressively degraded, the high texture frequencies are the first to be destroyed, and the high depth frequencies are likewise not recoverable. However



(a) Cylindrical texture 512×512. (b) Depth map 64×64. (c) Textured 3D object.

Fig. 3. (a) Cylindrical texture, location of the depth sample points (circles) and associated triangulation of the geometry. In reality there are 64×64 sample points. There are not sample points on the right border since they are identified to those of the left border through the cylindrical topology. Top and bottom half-bands of the texture are discarded. (b) Depth sampled in cylindrical coordinates. White pixels represent 3D points distant from the cylindrical axis. (c) 3D object reconstructed and meshed according to the depth map.

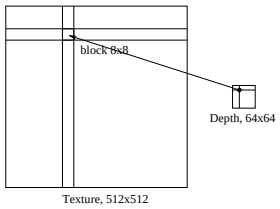


Fig. 4. Spatial coherence between payload and host information, by hiding each depth value in the corresponding 8×8 texture block.

the low depth frequencies remain recoverable. This achieves a progressive degradation of the payload when the cover is degraded.

III. Embedding algorithm

A. Overall hiding strategy

Using the previous spatio-frequential decomposition of depth and texture maps, we hide each depth coefficient of each wavelet band in the corresponding 8×8 block of the corresponding wavelet band of the texture (Fig. 5). Thus, one scalar value is hidden in a 8×8 block of scalar values. This is done by first quantizing the depth scalar value and then hiding each bit of the quantized depth value into one or more values of the 8×8 texture block, by modifying their LSB (least significant bit).

B. Quantization of depth

Each depth coefficient is quantized over a given number of bits. This number of bits as well as the quantization function (linear, logarithmic, etc.) and the quantization interval, must be carefully chosen, and may be different for coefficients of different wavelet bands. For our experiments we linearly quantized each depth coefficient using 8 bits. The quantization interval was empirically adapted to each wavelet band. Thus, each coefficient is represented as a binary number $b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0$.

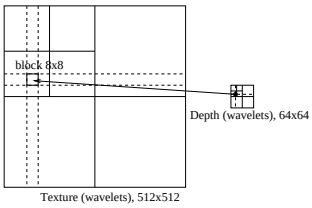


Fig. 5. Hiding each depth coefficient into the corresponding 8×8 texture block after wavelet decomposition.

C. Quantization of texture

A depth value is encoded in the corresponding 8×8 texture block. A bit of a depth value is encoded in a texture value by modifying its LSB. This LSB is defined unequivocally by a quantization step δ. Given this quantization step, the LSB b of a texture coefficient y is the LSB of the integer $n = \left\lfloor \frac{y}{\delta} \right\rfloor$.

Now, given a bit b_i to encode in the value y, we bring to y the least modification that sets its LSB to b_i in the most robust way. That is:

$$\hat{y} = \begin{cases} (n + \frac{1}{2})\delta & \text{if } b = b_i \\ (n + \frac{1}{2})\delta + \delta & \text{if } b \neq b_i \text{ and } y \geq (n + \frac{1}{2})\delta \\ (n + \frac{1}{2})\delta - \delta & \text{if } b \neq b_i \text{ and } y < (n + \frac{1}{2})\delta \end{cases} \quad (1)$$

Let us note that a different value for δ may be used for the different wavelet bands of the texture. Also, this set of values determines the visibility and robustness of the hidden data in the texture: the greater the δs, the greater the robustness, but unfortunately the greater the visibility too.

D. Repartition and redundancy of hidden bits

We have $8 \times 8 = 64$ texture values to encode one 8-bit depth value. Each texture value can encode one depth bit. So, there are 64 host bits for only 8 payload bits. This allows to redundantly encode some or all of the payload bits. For example we could encode 8 times each depth bit. However we choose to prioritize the most significant bits over the least significant ones. We hide 16 times bits b_7 and b_6, 8 times bits b_5 and b_4, and 4 times bits b_3, b_2, b_1 and b_0. By doing so, we hope that the most significant bits will be recovered more reliably using a majority vote.

This being set, we must still choose which host bits will encode which depth bits. For example we might encode 16 times the bit b_7 by modifying the LSB of a 4×4 sub-block of the 8×8 texture block. However many repartitions are possible. We actually used a repartition where each depth bit is more or less uniformly spread over the whole 8×8 texture block. This repartition, which gave the best results, is the following:

b_7	b_5	b_7	b_5	b_7	b_5	b_7	b_5
b_3	b_6	b_2	b_6	b_3	b_6	b_2	b_6
b_7	b_4	b_7	b_4	b_7	b_4	b_7	b_4
b_1	b_6	b_0	b_6	b_1	b_6	b_0	b_6
b_7	b_5	b_7	b_5	b_7	b_5	b_7	b_5
b_3	b_6	b_2	b_6	b_3	b_6	b_2	b_6
b_7	b_4	b_7	b_4	b_7	b_4	b_7	b_4
b_1	b_6	b_0	b_6	b_1	b_6	b_0	b_6

E. From embedding domain to RGB domain

After the depth information is hidden in the wavelet coefficients of the luminance, we perform an inverse wavelet transform to get the modified luminance map $\hat{Y}$. Then, the original RGB image must be modified to reflect the modification of luminance. Considering the simplified formula $Y = 0.1R + 0.6G + 0.3B$ for the luminance, we see that there are two degrees of freedom in choosing R, G and B to obtain a given luminance value: we can arbitrarily set two color components and still be able to get the wanted luminance by choosing the right value for the third component.

Let $d = \hat{Y} - Y$ be the difference of luminance introduced by the hidden data. The simplest way to modify the color components to reflect this change of luminance is to set $\hat{R} = R + d$, $\hat{G} = G + d$ and $\hat{B} = B + d$. However we must be aware that R, G and B must be integer. Therefore the increment d should be truncated to an integer value. In this way the luminance change is encoded with a precision of 1. However it is possible to encode d with a precision of 0.1 by using the two aforementionned degrees of freedom available when modifying R, G and B to get a given luminance.

Without loss of generally we can suppose $d \in [-0.5, 0.5[$ (by first adding the integer part of d to R, G and B). Then, to encode the fractional part of d we simply increase or decrease R, G and/or B by one as follows:

interval	coded	modification
$d \in [-0.5, -0.45[$	-0.5	$(R,G,B)+ = (0,-1,+1)$
$d \in [-0.45, -0.35[$	-0.4	$(R,G,B)+ = (-1,0,-1)$
$d \in [-0.35, -0.25[$	-0.3	$(R,G,B)+ = (-1,0,0)$
$d \in [-0.25, -0.15[$	-0.2	$(R,G,B)+ = (-1,0,+1)$
$d \in [-0.15, -0.05[$	-0.1	$(R,G,B)+ = (0,0,-1)$
$d \in [-0.05, 0.05[$	0	$(R,G,B)+ = (0,0,0)$
$d \in [0.05, 0.15[$	0.1	$(R,G,B)+ = (0,0,+1)$
$d \in [0.15, 0.25[$	0.2	$(R,G,B)+ = (+1,0,-1)$
$d \in [0.25, 0.35[$	0.3	$(R,G,B)+ = (+1,0,0)$
$d \in [0.35, 0.45[$	0.4	$(R,G,B)+ = (+1,0,+1)$
$d \in [0.45, 0.5[$	0.5	$(R,G,B)+ = (0,+1,-1)$

This modification has a minimal impact on the texture color and improves the precision with which the hidden information is finally coded.

IV. Extraction algorithm

A. Extraction and error correction

The first step of the extraction algorithm is straightforward: we simply perform the wavelet decomposition of the luminance of the texture image and we read the LSB of the wavelet coefficients. For a given 8×8 luminance block, each of the 64 extracted LSB corresponds to a given bit (from b_7 to b_0) of the associated depth value.

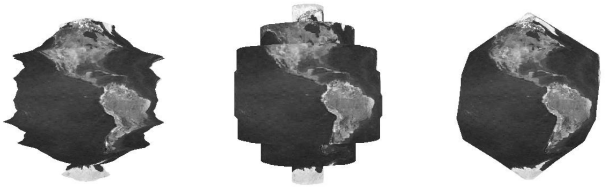
The second step is less straightforward: each read bit represents a given bit of a given depth coefficient. Since there is redundancy, several read bits correspond to the same depth bit. Ideally, these extracted bits would all be equal. In practice, due to attacks (e.g. compression of the image) or to the limited precision with which the payload was hidden, they might not be all the same. In this instance some kind of majority vote is needed to determine the value of the extracted bit.

For instance, if we read 12 times a 1 for bit b_7 and 4 times a 0, we would assign the value 1 for bit b_7. Yet, we explored a more general formula whereby "the value of a bit is no longer binary". Let's say we read k times 1 for a bit b_i and $N - k$ times 0. We then set the bit b_i to be:

$$b_i = \begin{cases} \frac{1}{2} + \frac{1}{2}\left(\frac{2k-N}{N}\right)^{\alpha} & \text{if } 2k > N \\ \frac{1}{2} - \frac{1}{2}\left(\frac{N-2k}{N}\right)^{\alpha} & \text{if } 2k < N \\ \frac{1}{2} \text{ if } 2k = N \end{cases} \quad (2)$$

where α is a chosen parameter. When $\alpha = 0$ we get back the majority vote formula. For $\alpha = 1$, the value of a bit varies linearly between 0 and 1 with respect to the number of 1s that are read. For $\alpha = +\infty$, the bit is set to 0 or 1 only when there is absolute consensus, otherwise it is "ignored" (i.e. set to 0.5). Note that different values for α could be used for the different bits and the different wavelet bands. Other formulas could be tried as well.

Once all the "continuous bits" b_i of a given depth coefficient are computed, the depth coefficient is simply computed as $\sum_{i=0}^{7} 2^i b_i$ and then scaled to the correct range, which depends on the wavelet band it belongs to. Except for the base wavelet band, this range is centered around 0, therefore when all b_i are set to 0.5, the reconstructed value is equal to 0. In other words, when the bits are "undecided", the reconstructed frequency value is not really used (since it is 0).



(a) 5/3 biorthogonal wavelets. (b) Haar wavelets. (c) "Linear interpolation" wavelets.

Fig. 6. Geometry reconstructed only from the base wavelet band (8×8 coefficients) of the cylindrical depth map, using three different wavelets.

B. Depth reconstruction

After all depth wavelet coefficients are computed from the extracted bits, we perform an inverse wavelet transformation to obtain the depth map in spatial domain. Then the 3D geometry can be reconstructed and textured.

The kind of wavelets used for describing the geometry is important: we want that if the high frequency bands are missing (i.e. have mostly null values) then the geometry reconstructed from the low frequencies be as smooth as possible. Not all wavelets are suited to this, as shown on Fig. 6. Therefore we used "linear interpolation" wavelets to decompose the depth (illustred on Fig. 7). However we used the so-called 5/3 biorthogonal wavelets (one of the wavelets that will be implemented in JPEG 2000 [8]) to decompose the texture.

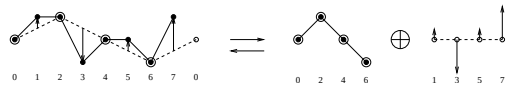


Fig. 7. Linear interpolation wavelets.

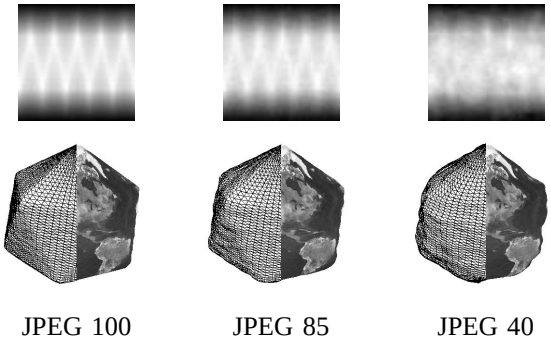


Fig. 8. Extracted geometry (with $\alpha = 1$) from the compressed texture: depth map (top) and corresponding 3D geometry (bottom).

V. Results

Fig. 8 shows the geometry that could be recovered after hiding the geometry into the texture and applying a JPEG compression (with $\alpha = 1$). Fig. 9 shows the numerical results of the distortion of both the texture image and the recovered geometry (for three different values of the reconstruction parameter α). The horizontal line represents the strength used for data-hiding.

As can be seen in this case, the formula used to handle the redundancy of hidden bits has an impact on the degradation of the recovered geometry. For slight JPEG compression (down to 50% quality factor) using $\alpha = 1$ gives better results than using a plain majority vote ($\alpha = 0$).

As for the property of spatial synchronization between payload and cover, it is illustrated by Fig. 10. We erased a small part of the texture. The redundancy of hidden bits allows then to detect that the data recovered in the corresponding area is not reliable: when we are "too far" from consensus for a given bit we simply declare it to be invalid instead of computing the depth value using an α-formula. When a depth value is invalid, it has a more or less localized impact according to which wavelet band it belongs to. In this example

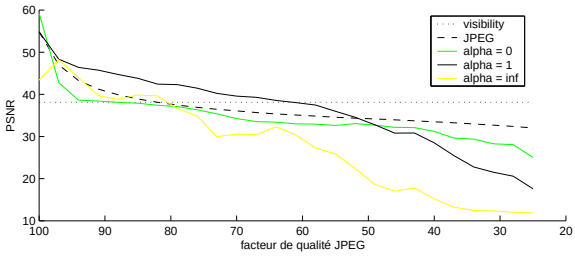


Fig. 9. Degradation of the hidden depth information in function of the lossy compression of the texture for three different values of the error recovery parameter α. The visibility of the hidden data (horizontal line) and the strength of the JPEG compression are also represented.

(a) Truncated texture. (b) Reconstructed geometry.

Fig. 10. Detection of unreliable parts of the texture. Spatial coherence between payload and host information.

we observe that small parts unrelated to the truncated texture part are also found to be invalid. This is because the blackened area affects the surrounding wavelets coefficients which may impair the data hidden therein. This confirms the need to use compact wavelets.

VI. Conclusion

We proposed a scheme for hiding a 2D depth map into a 2D texture map. As such this scheme is merely an image-in-image data-hiding algorithm. However in this instance there is a spatial correlation between the 2D images, which we wanted to preserve. Also, we wanted that the quality of the retrieved data be proportional to the quality of the host data. This was achieved to some degree by using a wavelet decomposition of both images.

Finally, such a scheme could be applied to any multi-band document where we want to reduce the number of bands by hiding the N-th band into the first (N-1) while ensuring the aforementioned properties. This might be useful to convey additional information through a standard data format that is not designed to encode this information. As for images it might then be possible to hide a transparency layer, or a stereo-vision disparity map, into a single image stored in a plain data format.

Acknowledgment

This research is supported in part by the FR-RNRT SEMANTIC-3D project.

References

[1] N.K. Adbulaziz and K.K. Pang. Robust data hiding for images. In *International Conference on Communication Technology*, 2000.

[2] J.J. Chae and B.S. Manjunath. Data hiding in video. In *IEEE International Conference on Image Processing*, 1999.

[3] J. Chou, K. Ramchamdran, D. Sachs, and D. Jones. Audio data hiding with application to surround sound. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2003.

[4] E. Garcia and J.-L. Dugelay. Texture-based watermarking of 3D video objects. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(8), August 2003.

[5] S. Katzenbeisser and F. A. P. Petitcolas. *Information Hiding Techniques for Steganography and Digital Watermarking*. Artech House, 2000.

[6] K. Solanki, O. Dabeer, B.S. Manjunath, U. Madhow, and S. Chandrasekaran. A joint source-channel coding scheme for image-in-image data hiding. In *IEEE International Conference on Image Processing*, pages II: 743–746, 2003.

[7] M. D. Swanson, B. Zhu, and A. H. Tewfik. Data hiding for video-in-video. In *IEEE International Conference on Image Processing*, volume 2, pages 676–679, Santa Barbara, CA, October 1997.

[8] D. S. Taubman and M. W. Marcellin. *JPEG 2000*. Kluwer Academic Publishers, 2002.