CORESA 2013 - 16ème édition du colloque COmpression et REprésentation des Signaux Audiovisuels
Le Creusot, FRANCE, 28 et 29 novembre 2013

Geometry-preserving Encryption for 3D Meshes

Marc Éluard, Yves Maetz, and Gwenaël Doërr
Technicolor R&D France

Abstract

This paper introduces a geometry-preserving protection paradigm that heavily distorts 3D objects while preserving some intrinsic geometrical property (e.g. the bounding box or the convex hull), thereby avoiding a global corruption of the whole 3D scene. Backward compatibility is guaranteed de facto : legacy non-compliant rendering engines can display 3D scenes containing protected objects, but only compliant renderers with the necessary credentials can display deprotected objects. We propose a couple of permutation-based encryption algorithms embracing this geometry-preserving strategy and then detail some challenges related to them. For instance, we discuss the side-effects of such protection mechanisms on other baseline 3D primitives, e.g. the rendering time, as well as the difficulty of assessing the security of such algorithms.

1 Introduction

The development of computer-generated graphics has been steadily increasing over the last decade. Today, virtual objects are routinely used in a number of applications, including socializing metaverses, games, simulation tools, and user interfaces. Blockbuster animation movies inherently rely on synthesizing rich and complex 3D worlds and incorporating visual effects into movies is now considered the norm in the entertainment industry [1, 2].

Creating these virtual objects/environments is a lengthy artistic process and the resulting end-product holds significant value. Losing control over the dissemination of such assets may lead to a number of uncomfortable situations for major Hollywood studios :

– unfinished 3D assets that hit the Internet can generate bad press for the studio that created it ;
– 3D objects that somehow leaked out of the content production workflow can be reused in other uncontrolled virtual context, potentially harming royalties revenues ;
– stolen 3D models could be exploited as a template to manufacture derivative products, e.g. by using modern 3D printers.

This threat analysis calls for a content protection framework for computer-generated 3D graphics.

Content protection architectures conventionally include three main components. First, encryption techniques are applied to digital data in order to make it unusable to parties that do not hold the necessary credentials. Second, an access control framework is set in place to (i) attach usage rules to digital items in the form of a ticket/license/permit, (ii) assign credentials to the individuals, and (iii) enforce the rules of the ecosystem. Third, a forensic mechanism is deployed in order to be able to locate the source of a leak in the event copyrighted assets were to appear on unauthorized distribution networks. This final component typically relies on the combination of digital watermarking technology [3] and traitor tracing codes [4].

In this paper, we will solely focus on 3D objects encryption. Section 2 first reviews conventional encryption strategies and how they prove ill-fitted to protect 3D objects. We then introduce the notion of *geometry-preserving encryption* and illustrate this new paradigm with two simple permutation-based encryption algorithms. Section 3 subsequently investigates how much these two protection algorithms interact with other regular 3D processing primitives. In particular, we analyze their impact on the rendering time. Then, we elaborate in Section 4 on the dangers of such handcrafted protection techniques. In particular, we exemplify that redunancy naturally present in 3D objects representation can be exploited by an adversary to reconstruct protected objects. Eventually, Section 5 summarizes our findings and outlines the main challenges that remain to be addressed.

2 Visual Access Control

2.1 Limits of Conventional Encryption

From the perspective of a cipher, bits are simply bits. It does not matter if those bits represent audio, video, or 3D graphics. They will eventually be ciphered in the same way. As a result, the most straightforward strategy to encrypt a 3D object is simply to bulk encrypt the binary file that describes it. The issue with such a direct approach is that the underlying structure of the file is also lost at encryption time. Legacy rendering engines oblivious to encryption will therefore fail to parse the file which could lead to a critical crash of the attached player.

A workaround consists in preserving the structure of the 3D file and to bulk-encrypt the structural elements independently. This is typically the solution adopted for XML encryption [5]. This revised strategy also provides a finer granularity, e.g. one could decide to only encrypt a few objects in a complex 3D scene. Nonetheless, major shortcomings remain.

A renderer that does not have the necessary credentials to decrypt data may decide to display nothing at the location

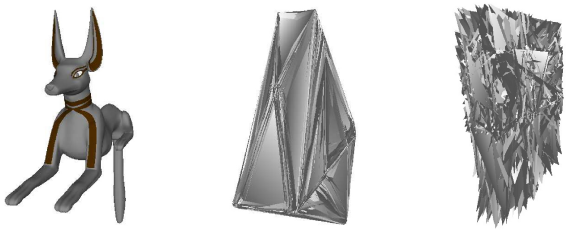


FIGURE 1 – *Illustration of the visual impact on the 3D object* Anubis *with the two proposed geometry preserving encryption techniques. While* `PointShuffling` *somehow exhibits a smooth surface that looks like a convex hull,* `CoordinateShuffling` *merely looks like a spiky object made of random triangles inscribed in the original bounding box.*

of the protected object. Depending on the targeted use case, it may not be the ideal solution. In a metaverse, if the user has no means to notice that 'something' is missing, it is unlikely that she will miss anything. The incentive to pay extra money to have access to enriched premium environments is lacking. In an animation production workflow, the studio may not be willing to give access to the full scene to a subcontractor. However, this subcontractor should be able to have access to visual cues in order to insert graphical elements without colliding with existing ones that would be invisible due to the protection framework.
Alternatively, the renderer could attempt interpreting bulk-encrypted structural elements like regular data and display protected objects accordingly. However, it would almost surely result in introducing a corrupted object that contaminates the whole 3D scene, hence nullifying any value the virtual world could have.

2.2 Geometry-preserving Encryption

In cryptography, the terminology *format-preserving encryption* refers to a type of ciphers that have the property of producing an output (the ciphertext) in the same format as the input (the plaintext) [6, 7]. The exact meaning of 'format' depends on the targeted application use case but, typically, only finite domains are considered. One could for instance want to guarantee that :

– encrypting a 16-digits credit card number remains a 16-digit number after encryption ;
– the encryption of a text made of English words results in another collection of English words ;
– an address composed of a number, a street, a postcode and a town is preserved by the encryption process.

The design of such ciphers was originally motivated by the need to impose legacy constraints for hard to maintain systems, e.g. financial transactions.
This prior work motivated our idea of defining a 'format' so that the encryption procedure preserves some geometric properties of the protected 3D object, hence coining the terminology *geometry-preserving encryption* (GPE). For instance, an encrypted object could be specified to remain with the bounding box (or the convex hull, or...) of the original 3D object. A direct by-product of such GPE is that it provides native backward-compatibility with legacy rendering engines. If a player cannot remove GPE protection (either because it is unauthorized to due to a lack of valid credentials, or because it is a legacy player that is simply unable to perform such operations) and still attempts rendering a virtual world composed of both genuine and protected 3D objects, the resulting displayed scene will look fine with the exception of the protected objects, which will remain unintelligible but contained within the geometric boundaries specified in the 'format'.

2.3 Two Illustrative Examples

A number of formats have been proposed to describe 3D scenes. They all rely on some kind of structure to separate different types of data. For instance, in the most simple case [1], a 3D object is composed of a set of V vertices with 3D coordinates $\mathbf{v}_i = (x_i, y_i, z_i)$ (aka. the *geometry*) and a set of F faces obtained by connecting the previously defined vertices with edges (aka. the *connectivity* or the *topology*). The structure of the file can be implicit through the use of a particular convention (e.g. OFF format), or explicit with XML-like tags to separate the different data elements (e.g. VRML and X3D formats).

`PointShuffling` (PS). A simple GPE strategy consists in shuffling the set of vertices of the 3D object by applying a key-seeded pseudo random permutation $\sigma_K()$ to the indices i of the vertices v_i. :

$$\forall i,\ \mathbf{v}_i \leftarrow \mathbf{v}_{\sigma_K(i)}, \tag{1}$$

where K is the secret key used to seed the pseudo-random number generator (PRNG). This process changes the set of vertices associated to a given face while leaving the 3D coordinates of the vertices untouched. Inherently, it is equivalent to pseudo-randomly rewiring the cloud of points defining the object. A PS-protected object is visually unintelligible and is contained within the convex hull of the original object as depicted in Figure 1. When having access to the secret key K, recovering the original object is then simply a matter of applying the inverse permutation to the set of vertices of the protected object.

`CoordinateShuffling` (CS). A variant of the previously described GPE strategy consists in shuffling independently the coordinates of the 3D vertices by applying a different key-seeded pseudo-random permutation to the indices i of the coordinates x_i, y_i, z_i of the vertices :

$$\forall i,\ \mathbf{v}_i = \begin{cases} x_i \leftarrow x_{\sigma_{K_x}(i)} \\ y_i \leftarrow y_{\sigma_{K_y}(i)} \\ z_i \leftarrow z_{\sigma_{K_z}(i)} \end{cases}, \tag{2}$$

where K_x, K_y and K_z are the secret keys (that could be derived from a single master key K) used to seed the three PRNGs associated to the axis of the 3D space. In contrast with PS protection, this process does create new vertices,

1. 3D objects could be further enriched with additional information e.g. normals, texture maps, normal maps, temporal animation information, etc. For the sake of simplicity, we will only consider objects made of vertices and faces in the remainder of this paper.

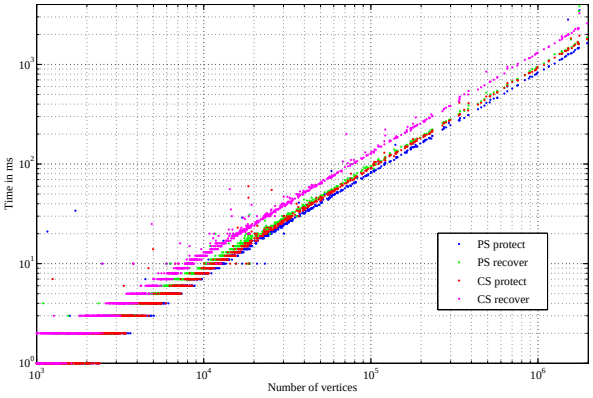


FIGURE 2 – *Protection/recovery time (in ms) of our two GPE algorithms vs. the number of vertices that the 3D object is made of. Beware of the log-log axis.*

although they remain in the bounding box of the original 3D object due to the specific way they are generated. The protected object now fills the whole bounding box including portions that are outside the convex hull of the original object as illustrated in Figure 1. When having access to the three secret keys K_x, K_y and K_z, recovering the original object is then simply a matter of applying the associated inverse permutations to the coordinates of the vertices of the protected object.

2.4 Computational Complexity

If the protection of a 3D object can be performed offline, the recovery mechanism will most likely need to be done online e.g. when the virtual world is loaded prior to rendering. Introducing delays in a creative workflow is the best recipe to have a technical solution ignored, regardless of its technical performances. As a result, we implemented the two proposed algorithms as a standalone application in order to measure the overhead introduced by the deprotection process. Figure 2 reports the protection/recovery times measured on a collection of over 2,200 3D objects of various complexity collected on the Internet.

We used the most straightforward implementation to perform permutations i.e. we loop through all the indices of the set of elements to permute and swap at each iteration the associated element with another one given by a call to a simple key-seeded PRNG. As a result, the protection/recovery time is expected to scale linearly with the complexity of the object and this intuition is clearly verified in Figure 2. In addition, the recovery process requires the swap operations to be performed in the reverse order than the one used for protection. From an implementation perspective, it implies that a first pass is needed to build the inverse permutation and a second one to apply it to the elements to be reordered. As a result, the recovery process is more 'expensive' than protection, and Figure 2 even shows that this difference is stronger for `CoordinateShuffling` than for `PointShuffling`. Finally, CS protection needs 3 times more calls to the PRNG than PS protection and it could therefore be expected for this complexity to translate in extra time cost in the same proportion. Figure 2 depicts that the ratio of the time cost between CS and PS is actually much lower, hence indicating read/write operations in memory (which are the same for both algorithms) are more prominent in our implementation compared to the calls to the PRNG.

In any case, the cost of the two proposed GPE algorithms is relatively small, with in average 1 second spent to permute 1,000,000 vertices. In other words, even for large 3D objects, the protection mechanism would only introduce a delay of a handful of seconds when loading an object which should not preclude their use in practice.

3 GPE and Legacy 3D primitives

The impact of introducing GPE in a 3D workflow should not be reduced to the time delays required to perform protection/recovery operations. Since GPE significantly alter the 'nature' of a 3D object, it has the potential to interact with legacy 3D primitives in a hard to predict fashion that could result in undesirable side-effects. In the remainder of this paper, we will focus on the interplay between GPE and the rasterisation process and highlight the consequences in terms of rendering time and frame rate.

3.1 Rasterisation Process

To display a 3D scene, the rendering engine goes through a process referred to as *rasterisation* [8, 9]. It essentially consists in identifying the position of the facets in the video frame captured by the virtual camera, and drawing the resulting transformed facets when they are not occluded.

To know whether a new incoming facets is occluded, the rendering engine maintains a z-buffer which records for each pixel the depth of the facet currently drawn in the frame. If for a given image pixel, the distance of the incoming facet is smaller than the one currently registered in the z-buffer, the considered pixel is updated using the current facet, as well as the associated entry in the z-buffer. Otherwise, it is discarded.

On its side, GPE distorts the original 3D object into something that may no longer *look like* a regular 3D mesh. For instance, the protected objects output by `PointShuffling` and `CoordinateShuffling` are made of facets that are in average much larger than originally. Displaying such objects will therefore require many more updates of the z-buffer than usually needed.

3.2 Metrics Definition and Validation

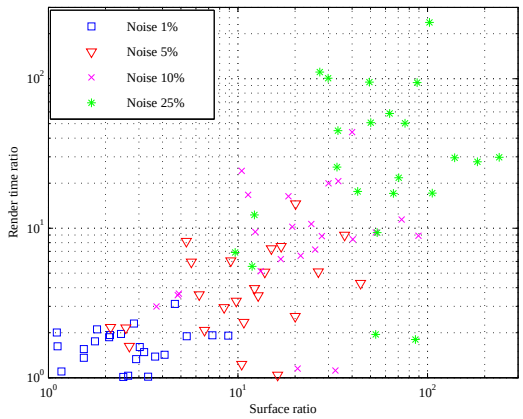
Measuring the impact of introducing a component on the rendering workflow would require instrumenting a 3D rendering engine in order to dump either the frame rate or the rendering time. This is however not an easy task in practice because of the lack of APIs in publicly available rendering engines. Moreover, it is usually quite tricky to isolate the actual rendering time from other processes which come into play e.g. the data transfer time from the hard

drive to the GPU. As a result, we decided to consider two very crude features that could be easily extracted from 3D objects in a systematic way and that permit to qualitatively infer the impact of a 3D geometry-processing primitive on rendering time.

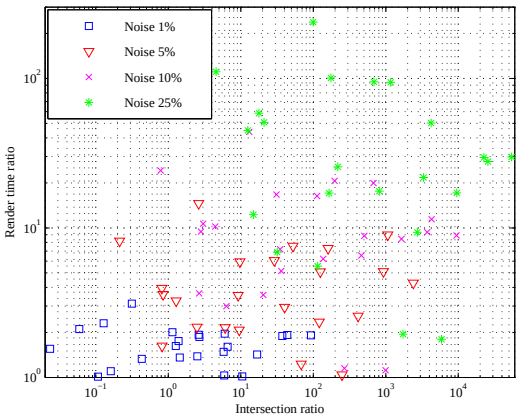
Average surface per facet. The rasterisation process involves *drawing* each facet that is not discarded by the z-buffer on the current frame. Intuitively, the larger the facet is, the more time it will require to be drawn. As a result, the first feature that we are considering is the average surface of the facets of the considered 3D objects. Increasing this quantity is expected to increase the rendering time.

Number of intersections. When the z-buffer is refreshed, it means that the facet corresponding to the values being overwritten has been drawn for nothing. In other words, the more overdraw there is when rendering an object, the more time is lost. The likelihood of such overdraw is somehow proportional to the number of facets hit when casting rays from the virtual camera. In order to be oblivious to the orientation of the camera, we count the average number of intersections when casting rays within the bounding box of the object using a fast segment-triangle intersection test [10]. We used 172 precomputed rays chosen to maximize the coverage of the bounding box. An increase of the number of intersections is anticipated to be an indication of an increased rendering time.

To validate these two metrics, we conducted a sanity check experiment where we compared them to the actual rendering time. The ground truth was obtained by manually recording for a few seconds the rendering time displayed in the control window of our proprietary renderer while the 3D object is rotating on itself and subsequently computing the average of the recorded rendering times. Due to the cumbersome aspect of such measurements, this validation was only performed on a few representative 3D objects.
Figure 3 illustrates the correlation of the two proposed metrics with the actual rendering time of a 3D object. In this case, we simply added some uniform noise to the vertices of the 3D objects and the amplitude of the noise is set with respect to the largest diagonal of the bounding box. We then measured (i) the rendering time, (ii) the average surface per facet, and (iii) the number of intersections in the bounding box. Eventually, we report the ratio before/after noise addition for each of the three metrics. In this setup, the rendering time appear to scale linearly with the average surface per facet whereas the relationship with the number of intersections in the bounding box seems to be more complex. Anyway, both figures clearly indicate that larger values of our two proposed metrics quantitatively corresponds to larger rendering times.



(a) Average surface per facet

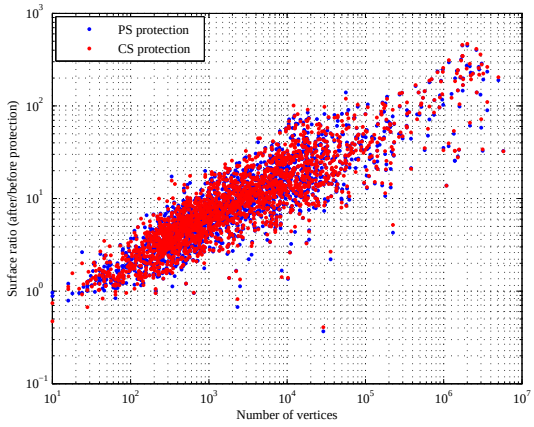


(b) Number of intersections

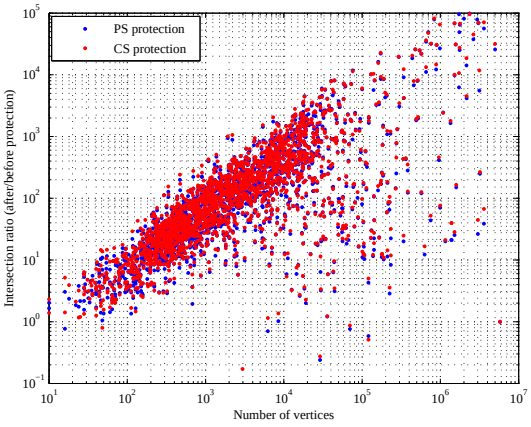
FIGURE 3 – *Sanity check experiment to validate the two crude metrics proposed to quantitatively assess the impact of inserting a 3D primitive on the 3D rendering pipeline. In this case, the 3D primitive is a simple uniform noise addition to vertices of the 3D object and whose amplitude is relative to the length of the largest diagonal of the bounding box.*

3.3 Experimental Results

Now that our two metrics are validated, we can benchmark our GPE algorithms, `PointShuffling` and `CoordinateShuffling`, in a systematic fashion to evaluate their impact on the 3D rendering pipeline. We measured the average surface per facet and the number of intersections in the bounding box for all the 3D objects of our database before and after protection. The experimental results are reported in Figure 4. PS and CS protection have a similar impact on the 3D rendering pipeline and it scales proportionally to the complexity of the 3D object. The more vertices has an object, the more impaired is the rendering time. In all cases, the influence of `Point-Shuffling` and `CoordinateShuffling` is significant, with values for our metrics on the high end compared to the ones obtained for noise addition in Figure 3.
All in all, it implies that our GPE algorithms have a significant impact on the 3D rendering pipeline and may well yield a much poorer frame rate or some kind of visual lag. One could of course argue that a user that does not have the necessary credentials to remove the protection should not complain if she gets a poorer visual experience. For instance, it may not be a scandal if a gamer experiences some lag when she enters a portion of the virtual world which she has not paid for to have access to. On the other hand, such



(a) Average surface per facet



(b) Number of intersections

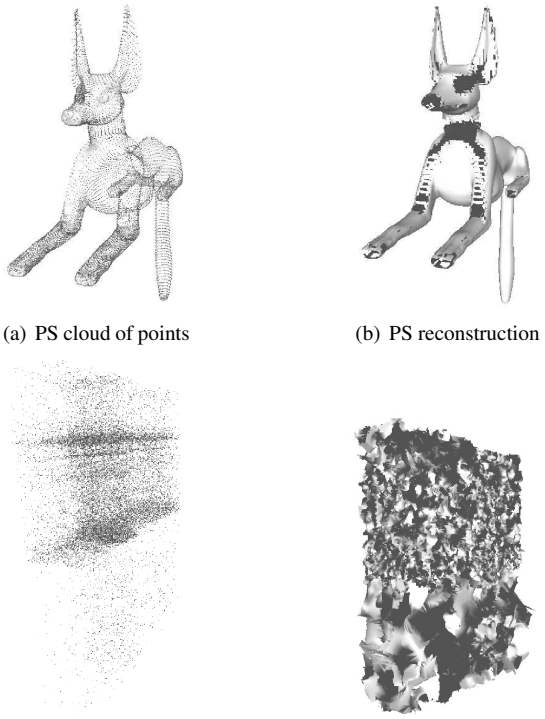
FIGURE 4 – *Impact of* `PointShuffling` *and* `CoordinateShuffling` *on the 3D rendering pipeline as measured using our proposed metrics.*

undesired side-effects may well preclude the introduction of these GPE algorithms on the 3D creative workflow. A graphist could accept not having access to the whole virtual world to do her job. However, having to deal with a constant degradation of her working conditions to respect this constraint is a no go.

4 Considerations on Security

Security requirements for the entertainment and the military industries are not the same. Still, the security of new ciphers proposed to protect multimedia items, including our GPE algorithms, needs to be properly evaluated to guarantee that an adversary cannot easily recover the unprotected items. Let us focus for the moment on `PointShuffling`.

One strategy to attack this algorithm is to attempt reversing the applied permutation. In general, such permutations are hard to reverse-engineer and the attacker is left with no other option than a brute force attack. Since there are $V!$ ways to permute V vertices and that 3D objects usually involve thousands or even millions of vertices, conducting



(a) PS cloud of points

(b) PS reconstruction

(c) CS cloud of points

(d) CS reconstruction

FIGURE 5 – *Illustration of the reconstruction 'attack' on the 3D object* Anubis *after* `PointShuffling` *and* `CoordinateShuffling`.

such an attack would require a prohibitive amount of time to be successfully carried out in practice. This being said, it would be naive to believe that the security level of PS protection is indeed $V!$. 3D objects naturally exhibit a strong structure. For instance, the vertices of a facet are generally not very far apart. Using such a priori information, it is imaginable to devise an efficient mechanism that would *guide* the selection of eligible permutations and discard irrelevant ones to speed up the search process. It is somewhat equivalent to reducing the key space.

Besides, `PointShuffling` suffers from a critical vulnerability. The elementary data unit onto which the permutation is applied is the set of coordinates defining a 3D point. As a result, no *new* point is created. This is a very useful property to keep control over the visual impact of the protection mechanism i.e. it preserves the convex hull of the object. On the other hand, it also implies that the underlying cloud of 3D points is not modified as depicted in Figure 5(a). Therefore, applying off-the-shelf surface reconstruction algorithms [11, 12] on this untampered cloud of points provides a good estimation of the original unprotected object as illustrated in Figure 5(b). Depending on the targeted application, such low-fidelity objects may be acceptable or useless. In contrast, since `CoordinateShuffling` applies a permutation independently on each coordinate, it is immune to such reconstruction as shown in Figure 5(d).

Nevertheless, it does not prove that CS protection is intrinsically secure. Surface reconstruction is only one out of

many yet unidentified threats against GPE schemes. The description of 3D object is very redundant, hence leaving room for information leakage that could be exploited by an adversary. For instance, should our basic 3D mesh be enriched with a texture, there is a strong correlation between (i) the distance between 3D vertices and (ii) the distance between the corresponding 2D points in the texture image. Scrambling a single one of these two elements leaves the door open to a potential reconstruction attack. Similarly, information about the normals of the facets also provides side information. The connectivity of the object provides a lot of information on the actual geometry of the mesh [13] and can thus helps to reconstruct an object. Security evaluation relying on the identification of potential threats due to information leakage will be one of the most sensitive tasks when designing novel GPE schemes for 3D graphics.

5 Conclusion

After detailing the limitations of conventional encryption techniques to protect 3D objects, we introduced in this paper the notion of *geometry preserving encryption* (GPE). In a nutshell, the baseline idea is to design a key-dependent reversible algorithm that would obfuscate the detailed semantics of the 3D objects while preserving some intrinsic geometric properties e.g. the convex hull or the bounding box. To illustrate this paradigm, we proposed two simple permutation-based encryption mechanisms.

We then discussed the side-effects of GPE on legacy 3D primitives with a focus on rendering time. Our experimental results clearly highlight that the introduction of GPE significantly impairs the 3D rendering pipeline, which could in turn affect the usefulness of such schemes in practical use cases. Such shortcomings call for new GPE algorithms which would leave a lighter footprint on the 3D ecosystem. However, the rendering time is not the only performance metric at risk. For instance, our GPE scheme may not interact very well with existing 3D streaming mechanisms. The renderer can no longer start displaying facets as they arrive on the fly since the 3D object needs to be fully received before being able to remove the protection. And it goes without mentioning potential optimization techniques that partition 3D objects in order to only send visible chunks to the renderer [14]. Existing compression schemes for 3D objects are also likely to be perturbed by GPE when they somehow rely on the same prediction framework as in audio and video [15], e.g. by representing a 3D mesh as an arrangement of baseline patches together with the associated prediction error. Depending on the target performance metric to be preserved, it is likely that alternate GPE strategies will be considered.

Eventually, we elaborated on the difficulty of assessing the security of GPE. It would be a mistake to reduce it to visual degradation of the object observed at rendering time and/or the security of the underlying cryptographic primitives. They are pre-requisites but they do not guarantee the overall security in any way. Multimedia contents in general, and 3D objects in particular, have a lot of redundancy and partial encryption may not provide the desired level of security, even if the content is unintelligible when it is rendered. Relevant side information may still leak due to natural redundancy in the signal that could help reconstructing the protected object to some extent e.g. by exploiting a prior on the statistics of the underlying signal. This is a lesson learnt the hard way in selective encryption for multimedia content [16], and we provided yet another example by pinpointing the conventional 3D reconstruction process that completely invalidates one of our GPE algorithms.

Bibliography

[1] Tim Wescott. The global animation industry : Facing the challenge of consolidation. Rapport technique, screendigest, November 2009.

[2] VegaH R&D Team. Animation – Market analysis. Rapport technique, VegaH Studios, January 2011.

[3] Patrice Rondao Alface and Benoit Macq. From 3D mesh data hiding to 3D shape blind and robust watermarking : A survey. *LNCS Transactions on Data Hiding and Multimedia Security II*, 4499 :91–115, 2007.

[4] Teddy Furon and Gwenaël Doërr. Tracing pirated content on the Internet : Unwinding Ariadne's thread. *IEEE Security & Privacy*, 8(5) :69–71, Sept.-Oct. 2010.

[5] Takeshi Imamura, Blair Dillaway, and Ed Simon. XML encryption syntax and processing. Technical report, W3C, 2002.

[6] John Black and Phillip Rogaway. Ciphers with arbitrary finite domains. In *Proceedings of of the The Cryptographer's Track at the RSA Conference on Topics in Cryptology*, volume 2271 of *Lecture Notes in Computer Science*, pages 114–130, 2002.

[7] Mihir Bellare, Thomas Ristenpart, Phillip Rogaway, and Till Stegers. Format-preserving encryption. In *Proceedings of Selected Areas in Cryptography*, volume 5867 of *Lecture Notes in Computer Science*, pages 295–312, 2009.

[8] James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes, and Richard L. Phillips. *Introduction to Computer Graphics*. Addison-Wesley Professional, August 1993.

[9] Carsten Dachsbacher, Philipp Slusallek, Tomas Davidovic, Thomas Engelhardt, Mike Phillips, and Iliyan Georgiev. 3D rasterization – Unifying rasterization and ray casting. Technical report, VISUS/Saarland University, August 2009.

[10] Nick Chirkov. Fast 3D line segment-triangle intersection test. *Journal of Graphics, GPU, and Game Tools*, 10(3) :13–18, 2005.

[11] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. In *Proceedings of the ACM SIGGRAPH Annual Conference on Computer Graphics and Interactive Techniques*, pages 71–78, July 1992.

[12] Robert Mencl and Heinrich Müller. Graph-based surface reconstruction using structures in scattered point sets. In *Proceedings of IEEE Computer Graphics International*, pages 298–311, June 1998.

[13] Martin Isenburg, Stefan Gumhold, and Craig Gotsman. Connectivity shapes. In *Proceedings of the IEEE Conference on Visualization*, pages 135–142, October 2001.

[14] Sheng Yang, Chang-Su Kim, and C.-C. Jay Kuo. A progressive view-dependent technique for interactive 3D mesh transmission. *IEEE Transactions on Circuits and Systems for Video Technology*, 14(11) :1249–1264, November 2004.

[15] Jingliang Peng, Chang-Su Kim, and C.-C. Jay Kuo. Technologies for 3D mesh compression : A survey. *Journal of Visual Communication and Image Representation*, 16(6) :688–733, December 2005.

[16] Amir Said. Measuring the strength of partial encryption schemes. In *Proccedings of the IEEE International Conference on Image Processing*, volume II, pages 1126–1129, September 2005.