

3D MODEL VISUALIZATION ENHANCEMENTS IN REAL-TIME GAME ENGINES

A. Merlo ^a, C. Sánchez Belenguer ^b, E. Vendrell Vidal ^b, F. Fantini ^a, A. Aliperta ^a

^a DIDA: Dipartimento di Architettura, Università degli Studi di Firenze, Italy -
alessandro.merlo@unifi.it; filippofantini@quipo.it; andrea.aliperta@gmail.com

^b Instituto de Automática e Informática Industrial - ai2, Universidad Politécnica de Valencia, Spain -
carsanbl@doctor.upv.es; even@upv.es

KEY WORDS: real-time simulations, displacement map, normal map, laser scanning, mesh processing, photogrammetry

ABSTRACT:

This paper describes two procedures used to disseminate tangible cultural heritage through real-time 3D simulations providing accurate-scientific representations. The main idea is to create simple geometries (with low-poly count) and apply two different texture maps to them: a normal map and a displacement map. There are two ways to achieve models that fit with normal or displacement maps: with the former (normal maps), the number of polygons in the reality-based model may be dramatically reduced by decimation algorithms and then normals may be calculated by rendering them to texture solutions (baking). With the latter, a LOD model is needed; its topology has to be quad-dominant for it to be converted to a good quality subdivision surface (with consistent tangency and curvature all over). The subdivision surface is constructed using methodologies for the construction of assets borrowed from character animation: these techniques have been recently implemented in many entertainment applications known as “retopology”. The normal map is used as usual, in order to shade the surface of the model in a realistic way. The displacement map is used to finish, in real-time, the flat faces of the object, by adding the geometric detail missing in the low-poly models. The accuracy of the resulting geometry is progressively refined based on the distance from the viewing point, so the result is like a continuous level of detail, the only difference being that there is no need to create different 3D models for one and the same object. All geometric detail is calculated in real-time according to the displacement map. This approach can be used in Unity, a real-time 3D engine originally designed for developing computer games. It provides a powerful rendering engine, fully integrated with a complete set of intuitive tools and rapid workflows that allow users to easily create interactive 3D contents. With the release of Unity 4.0, new rendering features have been added, including DirectX 11 support. Real-time tessellation is a technique that can be applied by using such technology. Since the displacement and the resulting geometry are calculated by the GPU, the time-based execution cost of this technique is very low.

1. INTRODUCCION

This paper is focused on the use of real-time and 3D modelling applications for the visualization and dissemination of the cultural heritage. In particular, the pipeline described modelling and texturing procedures developed for the conversion/optimization of dense point clouds into low-poly textured models that could be easily imported into game engines, while retaining the appearance of dense meshes from laser scanner surveys.

For this test, we decided to work on mascarón, a quite sizeable statue (measuring approximately 4 by 3 meters) of a mythological creature, found on the third of March 2009 in a grid of tunnels dug by looters at the base of the acropolis of Chilonché’s archaeological site (Guatemalan Petén, Figure 1), by an archaeological mission led by architect Gaspar Muñoz-Cosme of the Polytechnic University of Valencia and by the archaeologist Cristina Vidal-Lorenzo of the University of Valencia. Between 2010 and 2011, the research team accurately documented the archaeological finds through a traditional survey and a photographic campaign that produced 2D orthographic views. The location of the sculpture – only accessible through the grid of tunnels that surrounds it – and its complex morphology lends itself well to the use of 3D scanning techniques. So, in March 2012, the team, under the guidance of prof. Merlo who is experienced in digital surveys, set off on a campaign to use such technology to take the first experimental measurements (Figure 2). For an overall description of the organic shape, which a traditional 2D representation system could not easily provide (Figure 3), in May and June 2012 the

team decided to develop a mesh model of the sculpture, to be used for static renderings and real-time applications. The processing stage was supervised by Prof. Alessandro Merlo, Ph.D. Filippo Fantini, an expert in 3D modelling, and Dr in computer science Carlos Sanchez Belenguer.

To obtain high-resolution polygonal B-Reps from the initial point cloud, we followed a conventional pipeline (from Cyra Cyclone to Inus Rapidform). At the time of generating the mesh, we opted for the highest resolution provided by the



Figure 1. Archeological site of Chilonché in the region of Guatemalan Petén

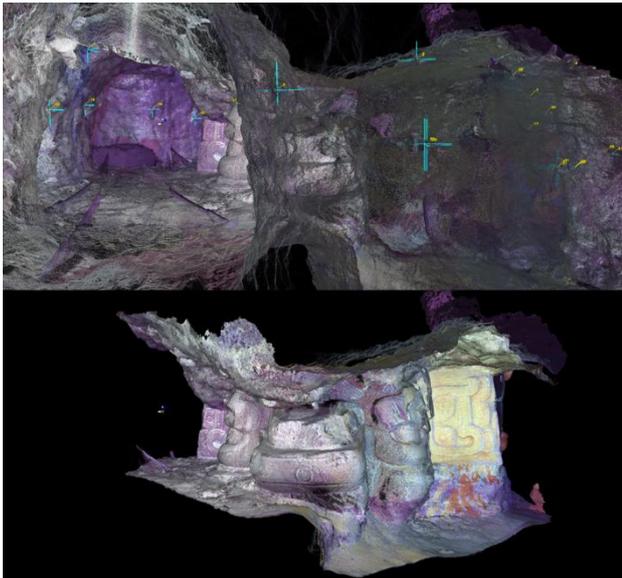


Figure 2. The tunnel dug from looters at the base of the acropolis

program instead of a preliminary mesh reduction, as we wanted to convert all geometrical detail into special bitmaps. Once the high-poly mesh had been obtained, we used a global remesh signal to correct for topological faults and create an isotropic mesh made of regular triangles.

At the first stage of research, the team chose to convert this high-poly model into an optimized one, focussing on classical approaches based on mesh density reduction (decimation), and to work with Low-Poly Models + Normal Maps to achieve a realistic representation, conveniently processed to perform within Unity 3D. The end result of this study was shown during the 17th International Conference on Cultural Heritage and New Technologies (11.06.2012, Vienna).

At the second stage, we decided to optimize the model using procedures borrowed from such entertainment applications as retopology and, with the support of architect Andrea Aliperta, other types of procedures: SubD + Displacement Maps. The release of Unity 4.0 actually added new rendering features that support real-time tessellation. In either case, our priority was to

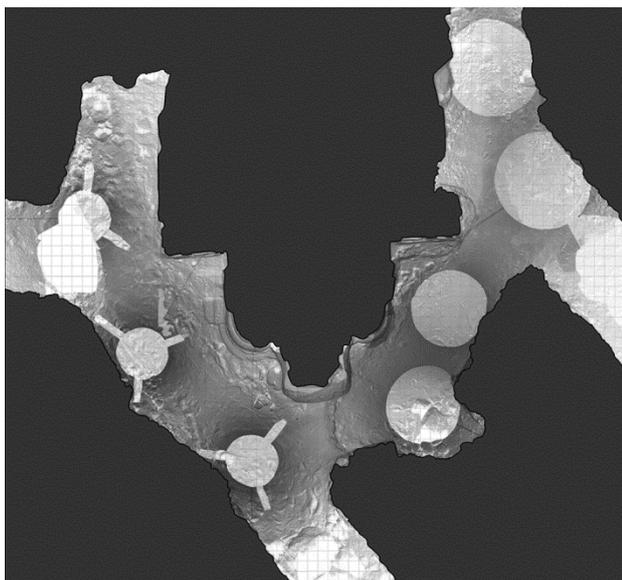


Figure 3. 2D orthographic view: horizontal section

monitor all steps in the “filtering” process, in particular the geometrical error introduced in the modelling process by the transition from laser scanner to game engine that, if not kept under control, may “damage” the metrical reliability of the meshes, so that, for example, interactive measurements and sections could not be taken right on the 3D scene.

2. PURPOSES

The main problem with real-time 3D simulations is duality between efficiency and realism, the former due to polygonal optimization, the latter to the quality of detail in the model. Until now, two of the most common solutions to this problem were using normal maps and different levels of detail (LOD) in models. The first solution consists in creating simple geometries and applying a texture, the normal map, on them, showing the normal vector of the surface in a specific point. With this information, flat objects can be shaded in a very realistic way without lots of polygons. The main problem with this technique is that flat faces remain flat, so, when the line of vision is parallel to the surface of the object, results are poor. The second solution consists in creating several 3D models for one and the same object, at different levels of detail. This way, when the object is far from the viewing point, a low-resolution model is displayed and, as the object becomes closer, more detailed models replace the low resolution ones. The main problem with this technique is that several models of one object need to be created, and all the levels of detail have to be loaded into the GPU in order to display the object. Real-time tessellation is a new approach that powerfully combines the two previous solutions together.

3. REVERSE-MODELLING AND MAPPING PROCEDURES

This case-study involves a comparison of two 3D model optimization procedures, both based on the use of bitmap images (hence the acronym IBGP, Image Based Geometric Processing), which transfer encoded colour and geometric information from a 3D digital model (generally high resolution) to another (usually low resolution) through special textures. This procedure takes inspiration from game applications, whose features (including those of the optimised 3D models) are borrowed from entertainment software.

This review covers different behaviours of normal maps and displacement maps, supported by a diffuse colour map applied to simplified versions of the same Hi-poly models.

As a first step, the simplified models had to be developed; the first one was developed through the direct decimation of polygons in the reverse-modelling software (Inus Rapidform, “decimate” command) and its surface is a knitted isotropic triangular (number of polygons 163494, average length of edges 19,41 mm); the second one was built instead through the so-called “retopology” technique. This consists in tracing the Hi-poly mesh of the model with a square mesh of lower density, known as “control mesh” (number of polygons 8016) or “quad-dominant mesh”. When drawing such mesh, the operator must constantly monitor the quadrangular polygon for regularity and the size of the edges so as not to build too sparse or too dense a mesh. In the former case, oversimplification would prevent any optimal geometric fitting on the background surface, resulting in a poor quality map. Small edges, however, force us to introduce a number of polygons that would make the entire procedure useless. It is also important that the mesh be built along the main directions on which the morphology of the object is based.



Figure 4. Different optimized mesh: (a) Triangular isotropic mesh. (b) Quad-dominant mesh. (c) Subdivision surface.

It is essential to dwell on the difference between the numbers of polygons in the two models. In the triangular mesh, the number of polygons is locked, while the quad-dominant model, because of its squared shape, can support conversions into Catmull-Clark-like subdivision surfaces; this makes it a variable-detail model, because the geometry of the mesh may be interactively increased by adjusting the level of subdivision (Figure 4).

Once we have the low-poly models, in order to encode the geometrical information and the texture colour, we need to create an UV map for each. This step involves associating the positions of the points of the 3D digital model within a reference system (u, v), in order to find an objective correlation between points on the plane and in space, so that the UV map can act as a unique reference system for every texture map that may be developed.

Importantly, the UV map must meet several important requirements. The most important one is that the polygons must not overlap, so as not to lose any correlation between the points in the model and those developed on the same plane. This requirement must be met when encoding both normal and displacement maps.

Another requirement that the UV map must fulfil, if concerning a control mesh, is that the polygon size in the model and the polygon size in the UV map must be directly proportional. If a polygon mesh Low-poly space approximates a defined number of polygons in the Hi-poly mesh, its image in the UV space must also be able to contain a reasonable number of pixels to receive geometrical information mainly through the baking process.

The baking process consists of an operation whereby the software stores information about the geometry and colour of the Hi-poly mesh, an RGB image or grey scale built on the UV reference system.

The geometrical information may be encoded in a normal map or in a displacement map; despite containing the same type of data, they are differently perceived.

The normal map is an RGB image and, using the tones generated by the three colour channels, it can enclose the trend of the Hi-poly surface normal. The colour of a pixel in the normal map is suggestive of the direction of a normal vector of the matching point, the surface of which has been baked. Once calculated and applied to the Low-poly mesh, the normal map simulates the behaviour of light reflections from the starting model and provides an image that is closer to the original non-decimated one: in other words, by adjusting the shading of the low-detail model, the normal map deceptively looks again like the original Hi-poly model.

The displacement map is, instead, a grey scale image, in which each shade of grey stands for a deviation of the background mesh from the optimized one. This is because, unlike the normal map, the displacement map, once applied to the model, displaces the points of the surface Low-poly based on the deviations calculated during the baking process. So, if you bake the surface of a displacement map, the UV map must also fulfil the requirement about continuity between adjacent polygons (Figure 5).

There are some criticalities in designing a texture displacement: the "pipeline" for the Save format, the features of the surface to

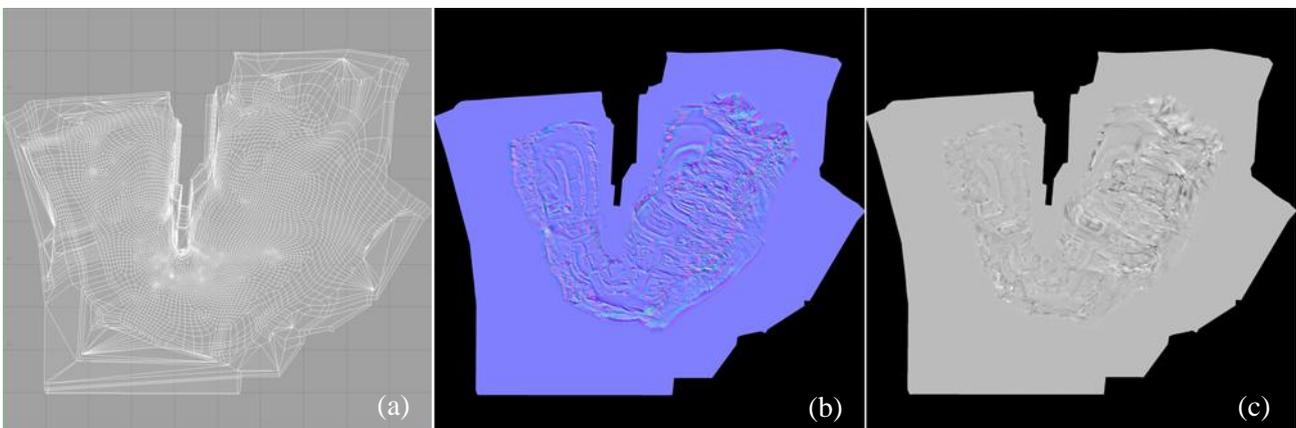


Figure 5. (a) The u,v reference system. (b) Normal map. (c) Displacement map.

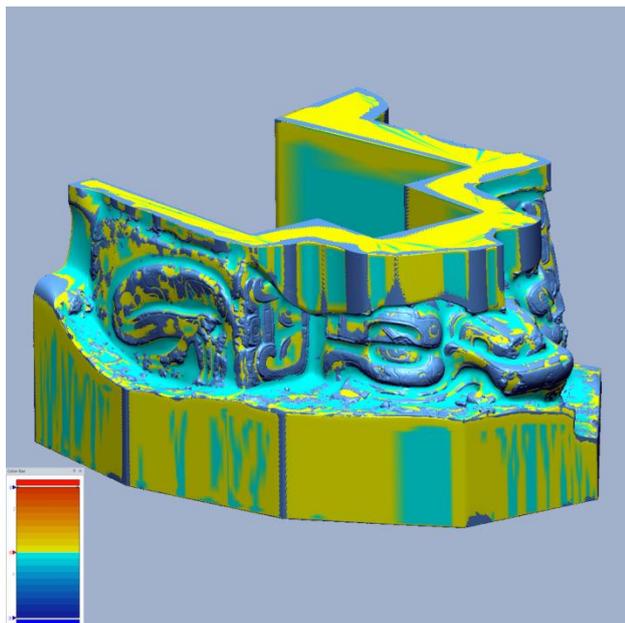


Figure 6. Deviation between Hi-poly mesh and Subdivision surface valued using Accuracy Analyzer.

be encoded, and the features of the subdivision surface it resembles.

Firstly, note that the software can produce and support a format with such a highly dynamic range as the OpenEXR. This format can be used to store information in 16 or 32 bits per pixel and provides over 255 shades of grey in an 8-bit image, for a more accurate rendering of the deviation of the points on the subdivision surface.

In this process, pride of place is given to the measurement of the distance of displacement; at the time of baking, the range within which the deviation between the two surfaces is to be measured must be defined. Such range is found by comparing the deviation between the two surfaces through the "accuracy analyzer" (a specific tool built into the reverse-modelling software) and is reusable after baking (Figure 6).

Essential to creating the colour texture is a proper photographic campaign to take photos that are perfectly focussed, with no dominant colour or shadow or line.

It was decided that an adequate amount of homologous points should be located on each image and Hi-poly model for the next

	Isotropic triangular mesh	Quad-dominant mesh
Number of polygon	163494	8016
Size of the file	16 Mb (.OBJ)	1,1 Mb (.OBJ)
Size of the normal map ¹ and displacement map ²	67,1 Mb (.TGA) (4069 x 4096)	5 Mb (openEXR) (4096 x 4096)
Size of the diffuse color map	19,2 Mb (.JPG) (8192 x 8192)	2,2 Mb (.JPG) (4096 x 4096)

Table 7.

camera re-sectioning process; high-contrast targets were placed on frames with the help of photo-editing software, while in Rapidform the control points on the Hi-poly model could be selected and exported to an .IGES format. These points were converted to a .DXF format, and the photos with the targets were imported into specific software (EOS Systems PhotoModeler), which could associate the homologous points and then locate the positions shown by the camera within the same reference model.

By exporting the camera to a .FBX format, it could be reused (in Luxology Modo) as a projector of its image on the surface of the model, and the result of such projection could be baked onto the UV map.

By repeating the same procedure for each frame until the model was fully covered and by blending together the images from each baking process, the diffuse colour texture was designed and eventually applied to the entire model by UV projection.

This procedure was only used for the triangular mesh model; to get the same texture as the one encoded in the UV map of the quad-dominant model, we used the "bake from object to render output" control, through which we could project the colour from the textured model onto the map of the UV texturized one. By enabling a "spread coefficient output" between the render settings, colour only could be baked, with no interference with the lighting of the model (Table 7, Figure 8).

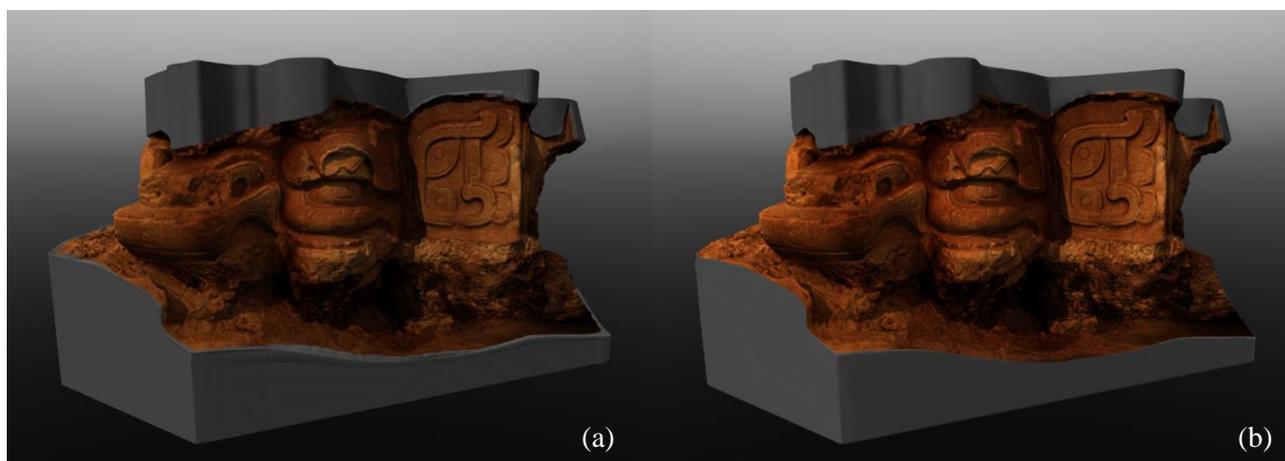


Figure 8. Comparison between optimized models. (a) Subdivision surface from quad-dominant mesh with displacement map and diffuse color map. (b) Triangular isotropic mesh with normal map and diffuse color map.

4. SHADER MODEL 5.0

Dealing with high-density 3D models in real-time visualizations has been traditionally complicated: the transfer of highly refined meshes between the CPU (*Central Processing Unit*) and the GPU (*Graphics Processing Unit*) has become one of the main bottlenecks of the rendering algorithm. Bandwidth limitations between both units, and memory restrictions impose very strict constraints to the real-time visualization techniques.

Prior to *Shader Model 4.0*, there was no possibility of doing per-primitive manipulation on the GPU, which means that models loaded in graphics memory could not be refined to more detailed ones.

In fact, only the *Vertex Shader Stage*, which is the first stage in the graphics pipeline, had access to vertex information, and the possibility to alter the original shape of the model. It performs per-vertex operations such as transformations, skinning, morphing, and per-vertex lighting. Its main disadvantage is that it always operates on a single input vertex and produces a single output vertex, so no refining operations are allowed.

The addition of a new pipeline stage in *Shader Model 4.0*, (Figure 9) called the *Geometry Shader*, unlocked new possibilities for programmers in creating procedural meshes on-the-fly. However, the limitation of primitives to refine geometry and its computational cost did not completely solve the problem of high-density 3D models visualization.

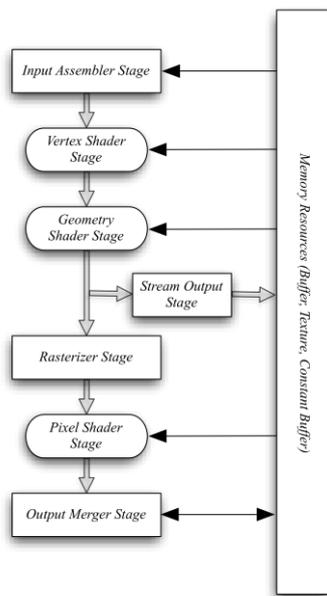


Figure 9. The graphics pipeline in *Shader Model 4.0*

With the release of *Shader Model 5.0* (Figure 10), three new stages have been added to the graphics pipeline: the *Hull Shader*, the *Tessellator* and the *Domain Shader*.

This new pipeline is able to create up to 8192 triangles for each triangle that receives, being the amount a parameter set by the programmer. Each one of these new triangles can be transformed programmatically on the GPU in a shader stage that grants access, not only to the input primitive vertices data, but also to the GPU generated triangle's vertices coordinates.

This way, *Shader Model 5.0* becomes an extremely efficient architecture tailored to create highly complex meshes on-the-fly and simple automatic *Level of Detail* algorithms. It provides a very high efficiency, since it trades *CPU-GPU* bandwidth for *GPU* arithmetic operations, which are rarely limiting during rendering.

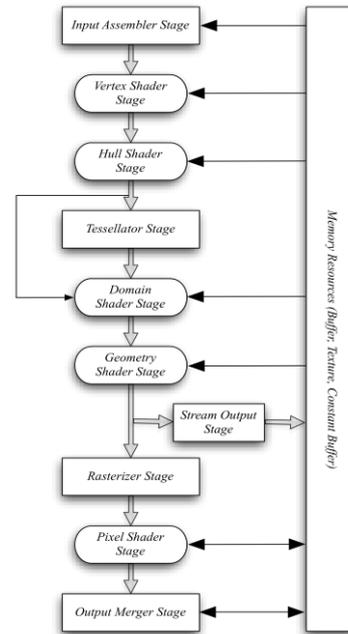


Figure 10. The graphics pipeline in *Shader Model 5.0*

The first stage of the new ones in the graphics pipeline is the *Hull Shader*. This one is a fully programmable stage that receives the output from the *Vertex Shader*. It is divided in two parts: the *Constant Hull Shader* and the *Main Hull Shader*. Both parts have access to the full primitive, and their purpose is to do any per-primitive calculations. The *Constant Hull Shader* is invoked once per triangle, and its goal is to establish how much the *Tessellator* has to subdivide it (*Tessellation Factor*). The *Main Hull Shader* is invoked once per vertex, and its purpose is to calculate any basis change on the primitive (control points). This way, the *Hull Shader* outputs the *Tessellation Factor* to the *Tessellator*, and the control points to the *Domain Shader*.

The *Tessellator* is the stage that deals with the triangle subdivision. It is not programmable and takes as input data the *Tessellation Factor* calculated in the *Constant Hull Shader* stage for each triangle in the low-polygon mesh. The result of this stage is a new set of triangles distributed in a regular grid, with texture coordinates UV varying from 0 to 1. These triangles are outputted to the *Domain Shader* with texture coordinates. Tessellation can be performed in two ways: integer and fractional. The first one consists in a simple symmetric subdivision, whilst the second one subdivides on the edges and morphs the vertices based on the decimal part of the *Tessellation Factor*. This way, when using a *LOD (Level of Detail)* scheme, mesh refining is smooth.

Finally, the *Domain Shader* is a fully programmable stage that has access to the primitive information, and gets invoked once per vertex generated in the *Tessellator*. It receives, as input data, the control points generated in the *Main Hull Shader* and the regular grid generated in the *Tessellator*. Its role in the graphics pipeline consists in calculating the position of the new generated vertices, converting their parametric UV coordinates into world space XYZ coordinates. Resulting triangles are passed to the *Pixel Shader* or if needed to the *Geometry Shader*. From here, the rendering pipeline continues as normal: the *Rasterizer Stage* converts the vector information (triangles or other primitives) into a raster image (a bitmap) and, the *Pixel Shader Stage*, invoked once per pixel by the *Rasterizer Stage*, deals with the calculation of the color values for each pixel, enabling rich shading techniques such as per-pixel lighting and post-processing.

5. ON-LINE REALTIME VISUALIZATION OF HIGH-DENSITY 3D MODELS WITH UNITY

Since the goal of this project is to disseminate tangible cultural heritage through real-time 3D simulations, we chose *Unity3D* as the rendering engine to do so. The main reason is because it allows to generate simulations for almost every device with enough graphics performance (PC and Mac computers, Android and iOS devices...) and, mainly, because it allows to display the simulations embedded in a web page, what we consider the best way to reach the maximum audience.

Since on-line visualization happens in a networked environment, the architecture of the application is based on a client-server paradigm with a Software Delivery strategy. There are four main components that play specific roles, and which are illustrated in Figure 10:

- Application server: stores 3D data, information and the application code, and transfers them to the clients. Since it serves to several clients, it does not provide complex computations for individuals. Instead of this, the server sends the application code and the data to the client, in order to generate the real-time simulation. Therefore, the computational power is provided by the end-user.
- Player server: stores the binary files that perform the web player installation on client computers. Since this web-application requires direct access to graphic hardware in order to perform a fast and complex simulation, clients have to install a player in their browsers that loads the code stored in the application server, and performs the simulation. Differentiating the application code and the player allows that, with only one installation, clients can load lots of different applications.
- Web client: in charge of the real-time rendering, interaction with the user, and all data requests to the server. First application run requires a player installation through the player server.
- Network: responsible for data transfers between client and server. Because of time delays, interactions between the user and the 3D environment have to be performed primarily on the client side, keeping this client-server communication only for data requests.

With the release of *Unity 4.0*, new rendering features have been added, including *DirectX 11* and *Shader Model 5.0* support. This makes *Unity* suitable for an advanced visualization like the one we propose.



Figure 12: Comparison between results achieved with simple lighting (left) and normal mapping (right). Notice how shading of surfaces improves by using normal maps, but how the geometric detail of the model remains constant.

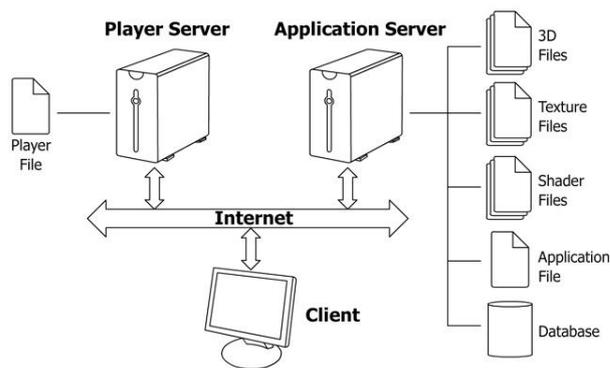


Figure 11. On-line application architecture.

5.1 Normal Mapping Technique

The first attempt to reduce the amount of polygons in the displayed model consists in using a low-poly mesh with a normal map texture to increase detail.

Normal mapping technique is very common in real-time 3D simulations, and it is used for faking the lighting on the surface of the model. With normal mapping, details can be added without using more polygons. A common use of this technique is to greatly enhance the appearance and details of a low polygon model.

Normal maps are commonly stored as regular RGB images where the RGB components corresponds to the X, Y, and Z coordinates, respectively, of the surface normal.

Since all normal mapping calculations happen in the *Pixel Shader Stage*, in the graphics pipeline, this technique does not have the capability to refine the model: the amount of vertices of the resulting model is always constant.

Results using this technique (Figure 12) are much better than results achieved with a simple lightning but the final 3D model is still to simple, and presents an important lack of geometric details.

5.2 Tessellation Technique

To prevent this from happening, we have introduced in the visualization a *Tessellation Shader* that, using the capabilities of *Shader Model 5.0*, takes advantage of the new rendering stages to, progressively, refine the model as the camera becomes closer to specific parts of the model.

The input data for the shader consists in a low-poly 3D model, a diffuse texture that stores the color information of the surface, and two special textures: a normal map and a displacement map. As explained before, these two textures have been calculated by projecting the original hi-poly model over the low-poly one, according to the normals of its polygons, and results are stored according to the UV coordinates of the low-poly model.

5.2.1 Hull Shader Stage: the proposed shader uses the *Constant Hull Shader* to calculate, for each triangle, how many times it has to be subdivided. This test can be based in two different criteria: the distance to the triangle, or the length (in screen space) of the edges of the triangle. The first one works good when all triangles of the model have a similar size, which is not the case. The second one, the one we are using, specifies the desired length, in screen space, of all edges in the final model. If an edge's length is greater than the desired length, it gets automatically subdivided.

This way, if the object is far from the viewpoint, original edges of the low-poly model will satisfy the subdivision criteria and, in consequence, they will not be subdivided, keeping the polygon count of the render very low.

As the viewpoint approaches the model, closer triangles will eventually fail the test, and thus, they will be refined into smaller ones.

Since all the information needed to calculate the position of the new vertices generated by the *Tessellator Stage* is included in the displacement map, there is no need to use the *Main Hull Shader* to calculate control points for the *Domain Shader Stage*. This stage is normally executed for alternative uses of the tessellation technique, as parametric surfaces rendering or smoothing algorithms.

5.2.2 Tessellation Stage: as said before, *Tessellation Stage* is not programmable, and it takes the original triangles with the *Tessellation Factor* calculated in the *Fixed Hull Shader* to perform the regular subdivision of each triangle. The only thing to specify in this stage is the way the *Tessellation Factor* has to be interpreted: as integer or fractional.

In order to prevent abrupt refinements of the model, tessellation is executed in *fractional* mode, which ensures that interpolation between different refinements is performed in a smooth way.

5.2.3 Domain Shader Stage: once the control flow reaches the *Domain Shader Stage*, all original triangles have already been subdivided, and it is necessary to establish the position, in world space, of each new vertex.

To do so, since *Shader Model 5.0* grants access to the primitive information in the *Domain Shader*, we calculate the corresponding pixel of the displacement map for each vertex.

The gray level of the pixel indicates how much the vertex has to be displaced, towards the normal of the face, so it fits the original high-poly model. By applying this transformation, we ensure that the deviation at this specific position of the refined model respect the original one is zero.

5.2.4 Pixel Shader Stage: since all geometric manipulation has been already performed, there is no need to execute the *Geometry Shader Stage*. Instead, the control flow of the rendering pipeline reaches its last stage: the *Pixel Shader Stage*. Here, the diffuse color of each pixel of the model is calculated according to the diffuse map, and the lightning is applied using the normal map texture.

It is important to remark that, for proper shading, the normal map texture becomes necessary: all new triangles generated by the tessellator are placed in their correct position, but they do not have a normal vector. If a normal map is missing, the final model geometry will be correct, but it will not shade properly.

5.3 Comparison between both techniques

Given that the tessellation shader includes in its last rendering stage the capabilities of the normal mapping shader, it can be said that it is more complete: the resulting shading of the model remains the same with both techniques but, with the tessellation shader, geometry gets progressively refined based on the viewpoint distance. This refinement improves, not only the precision of the model, but also visualization aspects that may affect to the shadow calculations, visibility culling... (Figure 13) This way, with the extra load of one displacement map, the same low-poly model used for normal mapping can exactly fit the high-poly one, and there is no need to load heavy meshes in the GPU. Also, since the refinement criterion is parametric, an infinite number of automatic LODs are displayed without any extra memory costs or modeling tasks.

6. CONCLUSIONS

In this paper, we have faced the challenge of cultural heritage dissemination through real-time 3D simulations. Since these simulations are intended to provide accurate-scientific representations, but also to be light enough for a standard computer to execute them, we have created a simulation that uses real-time tessellation techniques from *Shader Model 5.0*.



Figure 13. Comparison between the geometry of the original low-poly mesh (left) and the results of applying the proposed tessellation shader (right)

In order to perform real-time tessellation, the way 3D models and textures are created is a key factor. We have presented procedures in reverse modelling and mapping that allow generating low-polygon models from the original high-density mesh, and calculating texture maps that store all the data that has been lost during the simplification process: the normal map for the surface orientation, and the displacement map for the deviation between the hi-poly mesh and the low-poly one.

We have introduced the key aspects of *Shader Model 5.0*, that enabled us to implement a real-time tessellation shader and, with the model and textures created using the procedures explained in section 3, we have compared the results obtained with a traditional normal mapping shader and the proposed tessellation shader.

Given the results presented in section 5, tessellation shaders have proven to be more complete than normal mapping shaders. The main reason for this is that tessellation shaders can include in their *Pixel Shader Stage* the same code as normal mapping shaders but, only tessellation shaders can refine the low-poly model into more detailed ones.

This way, by parametrically refining the triangles of the model based on their distance to the viewpoint, we have achieved an extremely precise visualization, with a very low computing cost.

REFERENCES

- Alliez P., Ucelli G., Gotsman C., Attene M., 2008. Recent Advances in Remeshing of Surfaces. In: *Shape Analysis and Structuring*. Springer Berlin Heidelberg, Berlin, pp. 53-82.
- Apollonio F.L., Gaiani M., Manfredini A.M., 2010. Modellazione semantica, metodi a multirisoluzione. In: Benedetti B., Gaiani M., Remondino F. (editors). *Modelli digitali 3D in archeologia: il caso di Pompei*. Edizioni della Normale, Pisa, pp. 236-250.
- Brodie, K., 2000. Visualization over the World Wide Web. In: Scientific Visualization (Dagstuhl '97 proc.), *IEEE Computer Society Press*, pp.23-29.
- Cohen J., Olano M., Manocha D., 1998. Appearance-Preserving Simplification. In: *SIGGRAPH '98 Proceedings of the 25th annual conference on computer graphics and interactive techniques*. ACM New York, NY.
- Chittaro, L., Ranon, R., 2007. Adaptive 3D Web Sites. In: Brusilovsky, P., Kobsa, A., Nejdl, W. (editors), *The Adaptive Web - Methods and Strategies of Web Personalization, Lecture Notes in Computer Science*, Vol. 4321, pp. 433-464.
- Fantini F., 2010. Image Based Data Processing (IBDP): la restituzione attraverso displacement subD a partire dal rilevamento laser scanner. In: *Materia e Geometria*, 18/2010, pp. 149-158.
- Guidi G., Russo M., Beraldin J.A., 2009. *Acquisizione 3D e modellazione poligonale*. McGraw-Hill, Milano, pp. 294-300.
- Lai Y.K., Kobbelt L., Hu S.M., 2008. An Incremental Approach to Feature Aligned Quad Dominant Remeshing. In: *Proceeding SPM '08*. Proceedings of the 2008 ACM symposium on Solid and physical modelling, ACM New York, NY.
- Merlo A, Dalcò L., Fantini F., 2012. Game engine for cultural heritage. New opportunities in the relation between simplified models and database. In: Guidi G., Addison A.C. (editors), *Virtual Systems and Multimedia (VSMM)*. Proceedings VSMM 2012, Milano 2-5 settembre 2012, Institute of Electrical and Electronics Engineers, Inc., pp. 623-628.
- Merlo A., Vendrell-Vidal E., Fantini F., Sánchez-Belenguer C., 2012. Abstract "Mayan Mascarón from Chilonché (Petén, Guatemala): New Technologies for Cultural Heritage Dissemination". 17th CHNT International Conference on Cultural Heritage and New Technologies, Vienna, Austria, http://www.stadtarchaeologie.at/?page_id=5142.
- Papaoiouannou, G, et al, 2003. Enhancing Virtual Reality Walkthroughs of Archaeological Sites. In: *Proceedings of the Eurographics 4th International Symposium on Virtual Reality, Archaeology and Intelligent Cultural Heritage*, pp. 193-201.
- Verdiani G. (editor), 2011. *Il ritorno all'immagine, nuove procedure image based per il Cultural Heritage*. Lulu.com, pp. 11-14.
- Unity – Game Engine. <http://www.unity3d.com>. San Francisco, United States. (13 Jan. 2013)
- Zara, J., 2004. Virtual Reality and Cultural Heritage on the Web. In: *Proceedings of the 7th International Conference on computer Graphics and Artificial Intelligence*. Limoges, France, pp. 101-112.
- Zara, J., 2006. Web-Based Historical City Walks: Advances and Bottlenecks. In: *PRESENCE: Teleoperators and Virtual Environments*. Vol. 15, No. 3, pp. 262-277.

ACKNOWLEDGEMENT

This project has been partially financed by the "Programa de Ayudas de Investigación y Desarrollo (PAID)" of the *Universitat Politècnica de València* and the "Plan Nacional de I+D+i 2008-2011" from the *Ministerio de Economía y Competitividad*.