VIRTUAL TEXTURING IN THE DOCUMENTATION OF CULTURAL HERITAGE – THE DOMITILLA CATACOMB IN ROME

Irmengard MAYER[1], Claus SCHEIBLAUER[2], Albert Julian MAYER[2]

[1]Vienna University of Technology, Department of History of Architecture and Building Archaeology
Karlsplatz 13/251-1, 1040 Vienna, Austria

irmengard.mayer@tuwien.ac.at

[2]Vienna University of Technology, Institute of Computer Graphics and Algorithms
Favoritenstraße 9-11 / E186, 1040 Wien, Austria

scheiblauer@cg.tuwien.ac.at

mayer.julian@googlemail.com

Keywords 3D-modeling, visualization, texturing, out-of-core, photogrammetric reconstruction

Abstract

In the last decade the documentation of cultural heritage by means of laser range scanning and photogrammetric techniques has gained ever more importance. The amount of data collected by these means may be huge, and adequate presentation of 3D documented cultural heritage is still a challenge. For small and limited projects consisting of only a few range scans, the software provided with the laser scanner can be used for viewing and presenting the data. Large projects, consisting of hundreds of scan positions as well as projects where more and more data are collected over time, still have to deal with a massive reduction of the 3D data for presentation. Public demonstrations in museums, as for example shown by the Digital Michelangelo project, are already state of the art. The combination of huge point-base models and mesh models with high resolution textures in one viewer, the first type of models resulting from the data of laser range scans and the second type of models resulting from a photogrammetric reconstruction process, is still not available. Currently viewers are mostly limited to show models that are based on only one geometric primitive – either points or polygons – at once.

In the FWF funded START project "The Domitilla Catacomb in Rome. Archaeology, Architecture and Art History of a Late Roman Cemetery" – which is running for 5 years now – 3D point data was collected for the geometrical documentation of the vast gallery system of the Domitilla Catacomb, resulting in point data of some 2 billion (10^9) point samples. Furthermore high quality textured mesh models of the nearly 90 late Roman / early Christian paintings were generated with photogrammetric tools. In close cooperation with the Institute of Computer Graphics and Algorithms of the Vienna University of Technology the point cloud viewer Scanopy was improved for the combined presentation of huge point clouds and high quality textured mesh models in the same viewer. Our viewer is already capable of rendering huge point clouds, so for this a method to manage the vast amount of textures had to be found. Therefore we integrated a virtual texturing algorithm, which allows using the original photographs of the paintings taken on site to be mapped to the mesh models, resulting in a high quality texture for all mesh models. The photographs have a resolution of 11 Megapixels. Due to shortcomings in the programs used in the photogrammetric processing pipeline we scaled down the photographs to a 7.3 Megapixel resolution. Currently 608 of these images are used for texturing 29 mesh models. The work on the mesh models is still ongoing, and when all mesh models will be completed, we will use some 2000 images for texturing about 90 mesh models. These virtually textured models can show the details of each painting in the Domitilla Catacomb. When used in a virtual walkthrough the paintings in the catacomb can be presented to a broad audience under best lighting conditions, even the paintings normally not accessible by the public.

1. INTRODUCTION

The survey of huge cultural heritage objects was in the past a very exhausting work and in many cases not executable. The virtual presentation of such objects is still limited to relative simple reconstructions looking more like computer games than objects originating from reality. From the more than 60 late Roman-early Christian catacombs in Rome the bigger ones were never surveyed in their entirety due to lack of adequate surveying techniques. Archaeologically more interesting areas of these catacombs like family hypogea or painted tombs have been surveyed already very early (beginning with the end of the 16th century) with different methods but an overall survey was never made. With the upcoming of 3D laser scanners the geometrical survey of such objects became possible.

Since 2006 the vast gallery system of one of these catacombs, the Domitilla Catacomb, has been surveyed within the scope of the FWF funded interdisciplinary START-project "The Domitilla Catacomb in Rome. Archaeology, Architecture and Art History of a Late Roman Cemetery" (situated at the Austrian Academy of Science, Institute of the Study of Ancient Culture, project leader Dr. Norbert Zimmermann, financed by the Austrian Ministry of Education, Sciences and Infrastructure). Although the research history in different scientific disciplines reaches back 400 years the catacomb is insufficiently studied from a scientific point of view. Most importantly a complete documentation is missing, as this is a prerequisite for a scientific debate of modern methodical questions. On the base of a digital documentation a virtual model can be created. This model provides a frame for modern research questions like the reconstruction of the topographic development of the catacomb.

The Roman Domitilla Catacomb is located on Via Ardeatina in the South of Rome outside the ancient city walls. Beginning with some isolated pagan tombs in the 2nd century the Domitilla Catacomb grew in the 3rd and 4th century to a large Christian graveyard with the underground basilica of Nereus and Achilleus as a place of pilgrimage. The last written document about this catacomb dates back to the 9th century. In 1593 the Domitilla Catacomb was rediscovered by Antonio Bosio. He explored some regions of the catacomb and prepared the first documentation not only of the Domitilla Catacomb but also of other Roman catacombs [1]. In the following centuries several surveys were made of the catacomb but they all concentrated on areas with special interest - like an accumulation of paintings [1,2,3].

For surveying the geometry of the catacomb a Riegl LSM z420i [4] 3D laser scanner was used. For surveying the more than 80 late roman-early Christian paintings a Canon Eos 1Ds (20 and 14mm lenses) was used and photos were shot free-hand. To bring the laser scan data and the mesh models, which were created from the photos by a photogrammetric reconstruction process, into the same coordinate system, a Leica Total Station TCRM 1103i was used. The Total Station helped to measure a fix point net all over the catacomb. In 11 field campaigns about 2.000 scans were acquired of the more than 11km of galleries and more than 23.000 photos were captured for the generation of 3D textured mesh models.

1.1 Contribution

The contributions of this paper are:

- A workflow on how to get high resolution textured mesh models from free-hand taken photographs of (architectural) cultural heritage sites.
- An implementation of the "virtual texturing" technique for displaying these models in an interactive viewer in real-time. We describe the virtually texturing technique in this paper. We also use our implementation of virtual texturing [5] in a combined point and mesh viewer, called Scanopy.

With this viewer a virtual presentation of the ancient paintings of the Domitilla Catacomb with high textural details can be shown together with the point cloud model of the catacomb (consisting of some 2 billion points) on an off-the-shelf computer system. By using the virtual texturing technique the amount of data that can be handled for texturing the mesh models is not required to fit completely in the graphics or main memory of a computer, as it is managed out-of-core. This means that the complete data can be held on secondary storage devices, and only the parts required for rendering the current view are streamed to the memory of the computer.

2. PREVIOUS WORK

Texturing of mesh models is a standard technique to increase the quality of the visual appearance of polygon-based geometry by mapping an image or photograph onto the surface described by the polygons [6,7]. A requirement for texturing is a mapping of the vertices of the polygonal mesh to coordinates in the 2D image, and several well known methods exist to define a valid mapping, one example being “UV unwrapping”. When using several images for texturing, it is advantageous to compile a texture atlas from those images, and use this single image as source for texturing, as changing a texture binding during interactive visualization is a costly operation performance wise [8].

When dealing with large amounts of textures, the main memory of the computer might become too small to hold all texture data, preventing online access to textures when needed for rendering. Several techniques were developed which allow storing the complete texture data on disk and loading only the necessary data during rendering. The simplest approach is texture streaming, where textures are loaded on demand from disk. Dividing textures into separate tiles increases the granularity at which texture streaming can happen, and reduces the amount of wasted memory, first shown in terrain rendering [9]. This approach was enhanced by “Clipmapping”, a technique for real-time terrain rendering, where the separate levels of a complete mipmap chain (a mipmap chain is a collection of images of gradually reduced size, accompanying the original texture, which are used to speed up rendering and reduce aliasing artifacts), including the original texture, are each tiled and stored on disk, and depending on the current view position the necessary tiles are streamed to memory [10]. Another approach for real-time terrain texturing is to separately calculate the necessary mipmap level (and therefore the necessary texture tile) for each polygon that will be rendered, and not rely on the mipmap level calculation of the graphics card. This is done by estimating the size of a rendered polygon in screen space [11]. For generic geometry another level of indirection during texture coordinates calculation is necessary, the so called “pagetable”. With a pagetable the texture management much resembles the way how virtual memory is handled in an operating system, therefore it is also called virtual texturing (although this term is not always used consistently in literature, and it is sometimes also relating to different methods). Virtual texturing allows calculating the required mipmap level for the geometry per pixel, which was not feasible with previous methods. This can be done by rendering the geometry in a separate render pass into UV space [12] or into screen space [13], and afterwards stream the necessary tiles to the graphics card. Also the gaming industry makes use of virtual texturing, as mentioned by different game studios [13,14,15], to further increase the realism in state-of-the-art games.

Several tools exist for visualizing the data recorded by laser scans or created by photogrammetric reconstruction. Direct visualization of point-based data is often accomplished with viewers that come with the laser scanner or with specialized point rendering algorithms [16,17]. Since point data often becomes huge, it is beneficial to use out-of-core techniques and store the whole data on disk, while only rendering the currently visible part of the point data [18,19,20,21]. This technique was first shown on models captured by the Digital Michelangelo project [22]. The result of a photogrammetric reconstruction is a (sparse or dense sampled) point cloud which can then be converted to a mesh model and visualized with standard viewers like Blender [23], Maya [24], or MeshLab [25].

The combination of multi-resolution and multi-modal data is shown in [26,27], where huge data sets are collected from cultural heritage sites, which do not fit into the memory of an today’s off-the-shelf computer. In both cases the data had to be reduced for visualization, as the original data was too large and could not be handled for interactive visualization. In [28] a huge data set of Pompeii is prepared for interactive visualization, and the texture data had to be reduced for most textured object, as the full texture data could not be fitted into the memory of the graphics card. In contrast to these approaches, we are not required to reduce the point cloud data and the texture data and use the original data for interactive visualization, as we can handle data that does not fit completely into the memory of a computer. Only the geometric data of the mesh models is somewhat reduced, but since the exact geometric reconstruction was not a requirement for the visualization of the wall paintings of the catacomb, we deemed this limitation of our interactive visualization tool to be acceptable.

3. CREATING MODELS FROM PHOTOGRAPHS

The condition of the late Roman-early Christian paintings in the catacomb is in general very good. A survey by a 3D laser scanner cannot reproduce the quality of these paintings. With a photogrammetric survey of the

paintings it is possible to generate accurate three dimensional models with high quality textures. The experience showed that the visual quality of textured mesh models originating from laser scan data is not good enough. Especially the connection between geometry and texture in these models does not come up to the desired standards. By using photogrammetry the geometry develops out of the texture and discrepancies can be minimized.

To cover a whole 2m x 2m cubiculum with three arcosols (a standard situation in the catacomb) covered with paintings needs at least 18 positions of the camera. If the entrance gallery and the entrance wall should be modeled too, at least 12 more positions are necessary. By using a so called panoramic head, which allows the rotation of the camera around the focal point, photos of the entire cubiculum can be made from the same position. After capturing all images of one painting (for large paintings 700 or more images are necessary) they are loaded into the software CalibCam [29]. Here the registration of the images can be done semi-automatically. The exterior calibration of the camera positions can be calculated (using measured tiepoints from a total station is also implemented) as well as the interior calibration of the camera and the used lens. The lower the calculated error the better the following result of the textured models.

To create a mesh model one image pair showing the same area of the cubiculum respectively the painting is loaded into the software 3DM Analyst [29] and a textured mesh model is created. To cover a whole cubiculum many partial models have to be generated. Since these models overlap in many areas it is important to create a consistent and closed surface model. By exporting the vertices of each partial model from 3DM Analyst and importing them into Geomagic, a closed and, even more important, clean mesh model can be created [30,31]. To re-texture the model (color information is lost during the export) the images taken for the modeling process together with their orientation matrices are used. This work is done by the software 3DImage Software [32]. Figure 1 gives an overview of the complete workflow.

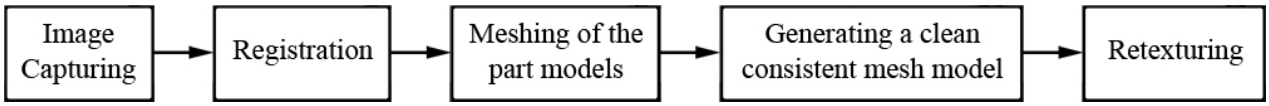

Figure 1: Workflow for the generation of textured mesh models.

Although for the capturing of the images special light panels are used, after the re-texturing process it is necessary to adjust the single images in their color and brightness to hide the seams where images are stitched together. After finishing this work the meshed and textured model can be imported into Scanopy without any further changes and can be displayed together with the point cloud of the Domitilla Catacomb where it will be automatically at the right position.

Since there are more than 80 paintings spread all over the catacomb, and for some textured models more than 60 images are used, it is essential to find a way to display all these models at once.

4. VIRTUAL TEXTURING OF MODELS

For rendering the mesh models of the cubiculae and wall paintings (see Section 4) in real time, we use a virtual texturing algorithm. In virtual texturing all images which shall be used for texturing are compiled into a single large texture, the “virtual texture”, which in this case resembles a texture atlas [8]. This texture is divided into regular tiles, which are then used for texturing instead of the original images. For the whole virtual texture a mipmap chain is generated, down to a size of one tile. The mipmap levels in this chain are numbered from 0 (for the level with the highest texture resolution, i.e. the resolution of the original images) to n (at the n-th level exactly one tile represents the complete virtual texture, at a low resolution). The size of a virtual texture is typically between $32K^2$ and $256K^2$ texels. Usually the pixels of an image are referred to as texels, if the image is used for texturing. The virtual texture and the whole mipmap chain reside on disk, and are referred to as the “tile store”. To use the virtual texture, a virtual texturing algorithm has to implement 3 main parts, which are executed in every rendered frame:

- Determination of virtual texture tiles that are needed for rendering for the current view point.
- Streaming these texture tiles into graphics memory.
- Map the texture from the loaded tiles to the geometry.

When creating the virtual texture, the texture coordinates of the geometry have to be adjusted so that they point to the right areas in the virtual texture. The buildup of the virtual texture and the offsetting of the geometry's texture coordinates are done offline in a pre-process. When the tiles are streamed to the graphics card, they are stored in the so called “physical texture”. The physical texture has a certain amount of free slots into which tiles can be loaded. During rendering this texture is bound for the fragment shader calls and used for all texture lookups of the geometry. Since the texture coordinates of the loaded tiles are not the same in the physical texture as they are in the virtual texture, a mapping between texture coordinates is necessary, and this is provided by the so called “pagetable texture”. In the pagetable texture each texel corresponds to exactly one tile of the virtual texture, including the complete mipmap chain. The pagetable texture stores the physical texture coordinates of the corresponding virtual texture tile. If a tile is not available in the physical texture, a fallback entry can be stored instead, e.g. the coordinates of the tile for the next available lower-resolution mipmap level. It is possible to hold the tiles of the highest levels (e.g. levels n, n-1, and n-2) always in the physical texture, so that when loading higher-resolution tiles these tiles can be used for rendering until the appropriate tiles have arrived. Note that the physical texture is not mipmapped, it only stores tiles of different mipmap levels of the virtual texture.

In the following the 3 main parts of the algorithm are described in more detail.

4.1 Determination of Virtual Texture Tiles

The determination of the virtual texture tiles is done per pixel in a pre-rendering pass. This exact tile determination in view space is possible by rendering the scene into the frame buffer, and storing in each pixel the information which tile coordinates are used for texturing this pixel, including the mipmap level. We use OpenGL as graphics API, and mipmap level selection is not defined in the OpenGL specification, it is left to the actual OpenGL implementation. Therefore one cannot always rely on calculating the mipmap level. Instead a separate texture lookup can be performed in the fragment shader that samples a mipmapped texture, which has stored the mipmap level in its texels. The result of this pre-rendering pass is an “image” in the frame buffer that contains in its pixels the addresses of all tiles that are necessary for texturing the currently visible geometry. Note that the tile addresses will not be unique, as a tile usually covers more than just one pixel in the scene. On the other hand this is even necessary for the virtual texturing algorithm to work, as it relies on spatial coherency for texturing the geometry.

When this pre-rendering pass is finished, the frame buffer is read back to the CPU. There the tiles for loading from disk are extracted from the buffer and the corresponding requests are sent to the tiles-loading thread.

4.2 Streaming Tiles into Memory and Updating the Physical Texture

Streaming the tiles from disk into memory happens after determining the tiles that are visible in the current frame. Streaming is done in a background thread, which loads the tiles asynchronously to the rendering thread. After loading, the tiles have to be decompressed, as they are stored in a compressed image format like JPEG or PNG. If desired the tiles can then be re-compressed to a format that is readable by the GPU, like the DXT compression formats. This reduces the memory requirements in graphics memory, but it also reduces the visual quality of the texture tiles. Decompression and re-compression can be done in a third thread.

The next step is to upload the newly loaded tiles into the physical texture on the graphics card. The physical texture is stored in a way that hardware filtering can occur when sampling the texture, i.e. as a 2D texture or as a texture array. Since current graphics drivers do not support enough layers for texture arrays to store enough tiles for a physical texture, we use per default a 2D texture, where the tiles are tightly packed in a

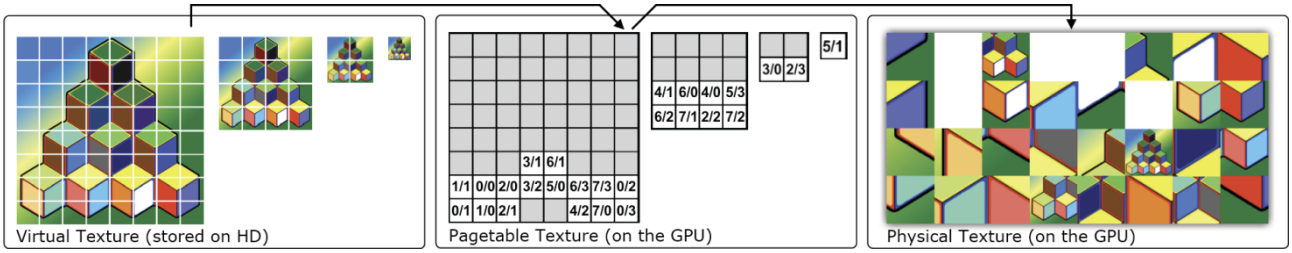

Figure 2: Overview of the mappings between the different textures that are used for rendering the virtual texture to the objects on screen.

grid layout. When there is no free position in the physical texture for the newly loaded tiles, then tiles can be replaced according to the “least recently used” paradigm [33], where those tiles, which have not been used for the longest time, are overwritten by the new tiles.

Finally, the pagetable texture has to be updated to reflect the changes in the physical texture. When a tile becomes available in the physical texture, its coordinates are stored in the corresponding texel in the pagetable texture. The fallback entries in the pagetable texture also have to be updated. Updates are done from the CPU, as the information which tiles are available in graphics memory, are also managed by the CPU. Figure 2 gives an overview of the mappings that occur between the textures that are used for managing the tiles.

4.3 Map Tiles onto Geometry

During rendering the virtual texture shader, a fragment shader, transforms the virtual texture coordinates of the geometry to coordinates in the physical texture, in which the tiles are stored on the graphics card. It does so by first sampling the pagetable texture to get the texel that holds the coordinates of the corresponding tile in the physical texture. All virtual texture coordinates that refer to one tile fetch the same texel from the pagetable texture. When the physical texture coordinates of the tile are known, the right offset within this tile is calculated and added to obtain the final physical texture coordinates, and this texture is then sampled at these coordinates to color the current pixel. Note that if the entry in the pagetable texture points to a fallback entry, the tile of the fallback entry will be used for texturing instead.

4.4 Further Considerations about Virtual Texturing

Although the size of a virtual texture is not limited per se, there are currently limitations due to restrictions of the graphics card hardware. One limiting factor is the length of the mipmap chain. Each additional level increases the size of the pagetable texture by a factor of 4. A mipmap chain of length 11 already results in a pagetable texture size of 1024^2 texels. For larger pagetable texture sizes a considerable performance hit occurs when updating the pagetable. Other limiting factors are the tile size and physical texture size. Since the physical texture stores all tiles that are needed for texturing the scene from the current viewpoint, the physical texture has to be large enough to provide space for the currently visible tiles. For a complex scene with many different sub-textures visible at the same time, it is advantageous to have many smaller tiles than only a few larger tiles. The tile size cannot be arbitrarily small (the side length of a tile is always an integer and a power of 2), as a decreasing tile size increases the number of I/O operations when loading the tiles from disk, and it also increases the size of the pagetable texture by a factor of 4. Therefore reasonable tile sizes are from 64^2 to 256^2 texels. The physical texture can be chosen as large as the maximum texture size that can be handled by the GPU, which is currently about 8192^2 texels. From these considerations results a maximum virtual texture size of $128K^2$ texels for the highest resolution level with a mipmap chain length of 11.

Creating virtual textures is a problem for common image manipulation programs like Photoshop[34] or Gimp[35], since these do not support images of $64K^2$ pixles or larger, as such images usually cannot even be imported or exported. Therefore the virtual texture has to be created in a distinct application, and cannot be edited interactively. Also viewing virtually textured models in common 3D-suites like Blender[23] or Maya[24] is not possible, since these programs do not support virtual texturing out of the box. We use a set of Python scripts to arrange the original images into a texture atlas, which is then divided into tiles, and all mipmap levels are created for these tiles.

5. RESULTS

A textured mesh model as reconstructed from the free-hand taken photographs can be seen in Figure 3. The model is displayed with a standard mesh viewer. For the reconstruction of the geometry of this model 296 photographs were used. The resulting geometry was then retextured with 32 of these photographs. These 32 photographs are then used for virtual texturing, although slightly downscaled (from 11 Megapixels to 7.3 Megapixels), as one of the programs in the model creating workflow can only handle quadratic textures.

We implemented the virtual texturing algorithm in a separate library [5], and integrated the virtual textured models via OpenSceneGraph [36] into the rendering pipeline of Scanopy.

We created a virtual texture of $128K^2$ texels resolution which holds all textures of the mesh models. For testing we used a computer with an Intel Q6600 CPU with 4 GB RAM, an NVIDIA 285 GTX graphics card with 1 GB VRAM, and 4 hard disks with 10,000 rpm in a RAID 0+1 configuration. In a walkthrough through the Domitilla Catacomb model with 1.9 billion points and 29 virtually textured mesh models the frame rate never drops below 10 frames-per-second, although we use 2 separate threads for loading, one for points and one for textures. The complete point cloud data uses 30 GB on disk (in about 260K files), and the virtual texture uses 6 GB on disk (in 1.4 million JPEG files). Figure 4 shows two different virtually textured models as they are rendered with Scanopy. The point cloud of the Domitilla Catacomb and the virtually textured models are displayed at the same time, as can be seen in the right image of Figure 4. When zooming into the virtually textured models, the fine details of the paintings on the wall become visible. In the left image of Figure 4 even the writings on the wall can be seen.

Figure 3: A mesh model photogrammetrically reconstructed and then retextured, displayed in MeshLab. This model is not rendered with virtual texturing.

Figure 4: These images show actual renderings of 2 areas of the Domitilla Catacomb displayed in Scanopy and using virtual texturing. In the right image the points from the point cloud can be seen in the distance.

6. CONCLUSION

We presented a workflow on how to create mesh models with high quality textures from photographs, and how to display such models on an off-the-shelf computer system. Therefore we implemented a virtual texturing algorithm, and imported the mesh models of the Domitilla Catacomb into our viewer Scanopy. It is possible to view point clouds and virtually textured models together in a scene in real-time, so users can get an impression of the catacomb's extension by navigating through the point cloud, and also view the wall paintings very closely and in great detail, which are textured directly with the photographs taken on-site. For future work we want to increase the budget on the polygons that we can handle in our tool.

7. REFERENCES

1. Bosio, A.: Roma Sotteranea, 1632.
2. De Rossi, G. B.: *La Roma sotterranea Cristiana*, 1864.
3. Krautheimer, R., et al.: *The Early Christian Basilicas of Rome*, Città del Vaticano, 1967.
4. Riegl Laser Measurement Systems, Horn, Austria: http://www.riegl.com/, 2011.
5. Mayer, A.J.: http://libvt.sourceforge.net/, 2011.

6. Catmull, E. E.: *A Subdivision Algorithm for Computer Display of Curved Surfaces*, PhD Thesis, Dept. of CS, U. of Utah, 1974.
7. Heckbert, P.: *Survey of Texture Mapping*, IEEE Computer Graphics and Applications, 6(11), pp. 56-67, November 1986.
8. NVIDIA Corporation: http://developer.nvidia.com/content/texture-atlas-whitepaper, 2011.
9. Cosman, M.: *Global terrain texture: Lowering the cost*, Proceedings of IMAGE VII Conference, 53–64, The IMAGE Society, 1994.
10. Tanner, C. C. et al.: *The Clipmap: A Virtual Mipmap*, Proceedings of the 25th annual conference on Computer graphics and interactive techniques, pp. 151-158, 1998.
11. Cline, D. et al.: *Interactive Display Of Very Large Textures*, Proceedings of the conference on Visualization '98, pp. 343-350, 1998.
12. Lefebvre, S. et al.: *Unified Texture Management for Arbitrary Meshes*, Technical Report RR5210-, INRIA, May 2004.
13. Mittring, M. et al.: *Advanced Virtual Texture Topics*, ACM SIGGRAPH 2008 classes, pp. 23-51, 2008.
14. Van Waveren, J.M.P.: *id Tech 5 Challenges - From Texture Virtualization to Massive Parallelization*, http://s09.idav.ucdavis.edu/talks/05-JP_id_Tech_5_Challenges.pdf, 2009.
15. Making Art Studios: *Lightning Engine - Virtual Texturing*, http://blog.makingartstudios.com/?tag=virtual_textures, 2008.
16. Botsch, M. et al.: *High-Quality Surface Splatting on Today's GPUs*, Symposium on Point-Based Graphics, pp. 17-24, Eurographics Association, 2005.
17. Dachsbacher, C. et al.: *Sequential Point Trees*, Proceedings of ACM SIGGRAPH, pp. 657-662, ACM Transactions on Graphics, 2003.
18. Scheiblauer, C. et al. : *Interactive Domitilla Catacomb Exploration*, VAST, pp. 65-72, Eurographics Association, 2009.
19. Rusinkiewicz, S. et al.: *QSplat: A Multiresolution Point Rendering System for Large Meshes*, Proceedings of the Computer Graphics Conference 2000 (SIGGRAPH-00), pp. 343-352, ACMPress, July 23-28 2000.
20. Gobbetti, E. et al.: *Layered Point Clouds*, Symposium on Point-Based Graphics, pp. 113-120, Eurographics Association, 2004.
21. Pajarola, R. et al.: Xsplat: External Memory Multiresolution Point Visualization. In Proceedings IASTED Invernational Conference on Visualization, Imaging and Image Processing, pp. 628–633, 2005.
22. Levoy, M.: *The Digital Michelangelo Project*, Proceedings of the 2nd international conference on 3-D digital imaging and modeling, pp. 2-11, 1999.
23. Blender: http://www.blender.org/, 2011.
24. Maya: http://usa.autodesk.com/maya/, 2011.
25. MeshLab: http://meshlab.sourceforge.net/ , 2011.
26. Guidi, G. et al.: *3D Modeling of Large and Complex Site Using Multi-sensor Integration and Multi-resolution Data*, VAST, pp. 85-92, Eurographics Association, 2008.
27. Remondino, F. et al.: *3D Modeling of Complex and Detailed Cultural Heritage Using Multi-Resolution Data*, Journal on Computing and Cultural Heritage (JOCCH), Volume 2 Issue 1, July 2009.
28. Pignatel, A. et al.: *Real-time Visualization of the Forum of Pompeii*, CAA Online Proceedings, March 22-29, 2009.
29. Adam Technology - 3D Measurement Software and Solutions: http://www.adamtech.com.au/, 2011.
30. Mayer, I.: *2d-texture builds 3d-geometry*, 13th International Congress Cultural Heritage and New Technologies, Vienna 2009, pp. 1-11, 2010.
31. Abdelhafiz, A., Mayer, I.: *Generating a photo realistic virtual model for the large Domitilla-Catacomb in Rome*, 9th Conference on Optical 3-D Measuremant Techniques, Vienna 2009, pp. 38-47, 2009.
32. 3DImage Software developed by Eng. Ahmed Abdelhafiz, Assuit Univeristy, Egyp, Civil engineering department.
33. Denning, P.J.: *The Working Set Model for Program Behaviour*, Communications of the ACM, pp. 323-333, Volume 11 Issue 5, May 1968.
34. Photoshop: http://www.photoshop.com/, 2011.
35. Gimp: http://www.gimp.org/, 2011.
36. OpenSceneGraph: http://www.openscenegraph.org/projects/osg, 2011.