EG UK Theory and Practice of Computer Graphics (2013)
Silvester Czanner and Wen Tang (Editors)

Acquisition, Representation and Rendering of Real-World Models using Polynomial Texture Maps in 3D

E. Vassallo[1], S. Spina[1] and K. Debattista[2]

[1] Department of Computer Science, University of Malta
[2] WMG, University of Warwick

Abstract

The ability to represent real-world objects digitally in a realistic manner is an indispensable tool for many applications. This paper proposes a method for acquiring, processing, representing, and rendering these digital representations. Acquisition can be divided into two processes: acquiring the 3D geometry of the object, and obtaining the texture and reflectance behaviour of the object. Our work explores the possibility of using Microsoft's Kinect sensor to acquire the 3D geometry, by registration of data captured from different viewpoints. The Kinect sensor itself is used to acquire texture and reflectance information which is represented using multiple Polynomial Texture Maps. We present processing pipelines for both geometry and texture, and finally our work examines how the acquired and processed geometry, texture, and reflectance behaviour information can be mapped together in 3D, allowing the user to view the object from different viewpoints while being able to interactively change light direction. Varying light direction uncovers details of the object which would not have been possible to observe using a single, fixed, light direction. This is useful in many scenarios, amongst which is the examination of cultural heritage artifacts with surface variations.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Digitizing and scanning I.3.3 [Computer Graphics]: Picture/Image Generation—Viewing Algorithms I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

1. Introduction

There is an increasing demand for ways to represent real-world objects digitally. Digital representation of objects can enable their examination by anyone from anywhere. However, there are several factors which are posing challenges in each of the three phases involved in the process.

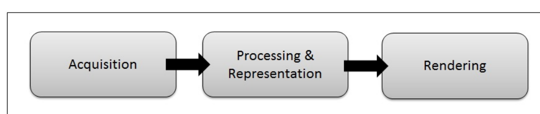

Figure 1: *Pipeline showing the three phases required to digitally represent a real-world object*

During the first phase, acquisition, the main challenges are the affordability of the hardware required to gather the data, the skill required to operate the hardware, and being able to capture the information on the site itself. The main challenges during the second phase, processing and representation, are the skill and time required to process the acquired data such that it is storage-ready, and then being able to store it using reasonable file sizes. Finally, during the rendering phase, the acquired and represented data needs to be presented to the user using realistic rendering with reasonable performance on commodity hardware.

Our work attempts to tackle these challenges. The contributions of this paper are outlined below:

- For the acquisition phase we propose a method in which 3D geometry, texture and reflectance behaviour can all be captured using only Microsoft's Kinect device, three coloured reflective spheres, and a light source.
- We propose a processing pipeline through which the acquired 3D geometry can be processed to obtain a model covering a 360° view of the object. Since this pipeline is almost entirely automatic, it can be carried out by anyone.

© The Eurographics Association 2013.

DOI: 10.2312/LocalChapterEvents.TPCG.TPCG13.077-084

delivered by EUROGRAPHICS DIGITAL LIBRARY
www.eg.org diglib.eg.org

- We propose a way to take advantage of overlapping Polynomial Texture Maps (PTMs), in order to add detail to a relatively coarse 3D model. This also contributes to alleviating the problem of large file sizes.
- Finally our work proposes a method for mapping the 3D geometry and the overlapping PTMs, and presenting the result in a viewer which would allow the observer to move the camera and change light direction to uncover object details. The proposed method can be programmed to be executed on the GPU in almost its entirety, greatly improving rendering performance.

2. Background and Previous Work

A method that is often used to capture 3D geometry is laser scanning. Although it is claimed that data gathered using this method is very precise [DCCS06], disadvantages of laser scanning include the cost of the equipment [MG02] and that the process requires demanding skill sets [MMSL06]. Our work provides a framework in which the Microsoft Kinect sensor, a relatively cheap device, can be used to capture the 3D geometry. In a previous project, the Kinect device has already been used as a 3D scanner [TFS*12]. Although results were satisfactory, besides the Kinect sensor this method requires an optical tracking system and four ceiling-mounted infra-red cameras. This leaves the challenges of hardware affordability and demanding skill sets untackled.

In another similar project, called KinectFusion, the Kinect sensor can be moved through space so that it creates 3D surface reconstructions of real-world environments in real-time [IKH*11]. While the sensor is moved, depth and RGB data are continuously read, as are the 6 degrees-of-freedom (6DOF) of the camera (forward/backward, up/down, left/right, pitch, yaw, roll). Using this information the depth data from the different viewpoints is fused, using the Iterative Closest Point algorithm. The colour information is used to build textures which are then mapped to the 3D reconstruction. Reflectance information however, is not being captured in this work.

Texture representation plays a big part in creating a realistic digital render. Using normal image files as textures (traditional textures) is usually not enough for realistic rendering of more complex materials [MMK03]. This is because the lighting conditions in which the texture would have been photographed/created, would have been embedded inside the texture [HOMG03]. If it is rendered in those same lighting conditions, it would look realistic, but this is hardly ever the case and precludes the use of the texture in other lighting scenarios, which is the primary goal of many rendering systems. Using bump maps, specular maps and/or normal maps along with traditional textures does allow relighting as light is moved around the object, making the surface look rough and bumpy when it is actually flat. However, creating maps, for example bump maps, of real-world textures from photographs is very difficult [MGW01]. Besides, the existing methods for this purpose are unable to capture effects resulting from large surface variations, such as self-shadowing and intra-object interreflections [MGW01].

A Polynomial Texture Map (PTM) is a type of texture in which each texel, instead of being defined as having a fixed colour value, is defined by a second-order bi-quadratic polynomial function [MGW01]. PTMs are created by capturing a set of photographs of an object in which the camera viewpoint is fixed but the light direction is varied. Therefore PTMs preserve self-shadowing and interreflection of light [MVSL05]. Creating the PTM requires the precise vector of the light direction in each photograph. A technique to extract the light direction after taking the photographs is Highlight Reflectance Transformation Imaging (HRTI). In this technique the object is photographed together with one or more glossy spheres around it. During processing, the light direction is recovered from the specular highlights produced on the sphere/s [MMSL06]. During rendering, the polynomial functions for each texel would yield different colour values for different light directions [MGW01].

At the time of writing, PTMs have mostly been treated as simply light-adjustable image files. However, in a previous experiment PTM textures were used to add detail to coarse, low-resolution 3D models [RN10]. However, in this work they do not capture texture and reflectance behaviour of the whole object, but rather capture a "patch", and then synthesise it using specific algorithms to make it cover the whole object. While this might simplify the capturing process, it will only create satisfactory results in cases where the object has a homogeneous texture. Inscriptions or other features that are not consistent along the surface of the object will not be represented correctly, questioning the suitability of such method for cultural heritage purposes.

c-h-i have introduced the concept of multi-view reflectance transformation imaging, which is a set of PTMs of the same object, captured from multiple camera viewpoints. Optical flow data makes it possible to view the PTMs as if they were one file [Cul10]. However, since PTMs are not being mapped to 3D geometry, allowing the user to move the camera would create unrealistic results.

3. Acquisition, Processing and Representation

3.1. 3D Geometry

The first phase in our pipeline is the acquisition of the shape of the object, i.e. the 3D geometry. As illustrated in Figure 2, in our proposed method the object is surrounded by three differently coloured reflective spheres, and placed at least 1m in front of any background/wall. The Kinect sensor is then positioned such that the object and the spheres are within the camera's viewport. Both depth and RGB data are read from the sensor. This process is repeated for a number of overlapping viewpoints around the object and spheres.

© The Eurographics Association 2013.

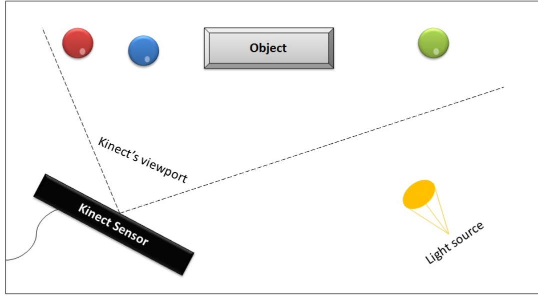

Figure 2: *The capturing setup*

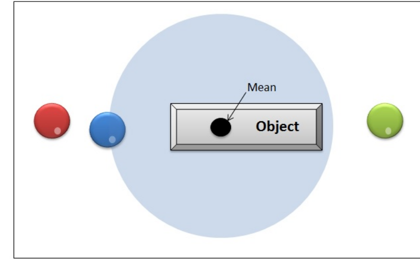

Figure 3: *The filtering process*

During processing the spheres will serve three purposes: filtering the object from background noise, registering scans from different viewpoints, and calculating light direction by detecting highlights.

The captured geometry is then processed through a pipeline as described below.

3.1.1. Sphere Detection and Adjustment

Since RGB data is read using the Kinect's camera, a circle detection algorithm can be used to automatically detect the spheres. Since the depth sensor and RGB camera are mounted in different positions on the device, the detected positions from the camera need to be projected to the equivalent positions in the depth map, using the equations below.

$$\begin{aligned} d' &= rawToWorld(d) \\ d_x &= ((x_c - cx_d) * d')/fx_d \\ d_y &= ((y_c - cy_d) * d')/fy_d \\ d_z &= d' \end{aligned} \tag{1}$$

where *rawToWorld* is a function which converts raw depth values (0-2048) to world metric space, x_c and y_c is the 2D coordinate detected from the RGB camera, d is the depth value at that particular 2D coordinate, and cx_d, fx_d, cy_d, fy_d are constants specific for every Kinect device which can be discovered using a calibration process.

3.1.2. Filtering

The sphere centres calculated from the previous phase are used to filter the object from any background noise. As illustrated in Figure 3, the mean position of the two spheres which are furthest from each other is calculated and only points which are close enough to the mean point are retained. The sphere centres are also used to remove the points representing the actual spheres.

The value which defines "close enough" might need to be changed for different objects. Also, our method implies that the object needs to be placed quite some distance away from any background. If the spheres were placed surrounding the object (i.e. forming a triangle around it), filtering would have been much more straightforward as it would have solved the two issues outlined above. However, placing them in that manner would mean that they will not all be visible from viewpoints around the object, complicating registration.

3.1.3. Registration

In this phase the vertices obtained from different viewpoints need to be registered into one complete point cloud. Since the three spheres are present in every scan, their 3D location in each scan is used to calculate the transformation matrices required for registration.

Registering point cloud A with point cloud B would require the following calculation:

$$B = RA + T \tag{2}$$

where R is the rotation matrix and T is the translation matrix.

The centroid of each set of 3 spheres (a set for each viewpoint), C_A and C_B, is calculated, and subtracted from each sphere in the respective point cloud. This eliminates translation, leaving only the rotation component. The covariance matrix is then calculated as follows:

$$Q = \sum_{i=1}^{N} (P_A^i - C_A)(P_B^i - C_B)^T. \tag{3}$$

Singular Value Decomposition (SVD) is then used to factorise matrix Q into U, S and V:

$$(U, S, V) = SVD(Q). \tag{4}$$

The rotation matrix is calculated using the V and U components resulting from SVD:

$$R = VU^T. \tag{5}$$

Finally, the translation matrix is calculated by subtracting the rotated centroid of cloud A from the centroid of cloud B.

$$T = C_B - RC_A. \tag{6}$$

© The Eurographics Association 2013.

3.1.4. Vertex Clustering

In this phase the amount of vertices in the point cloud are reduced while still retaining the approximate shape. Reducing the amount of points would yield certain advantages, such as make the rest of the processing pipeline more efficient, and allow for faster rendering. The method used is vertex clustering, in which the bounding space of the object is partitioned into cells, and then a representative vertex is computed for every cell. If $P1, P2, \ldots, Pk$ are points in the same cell, the representative point P for that cell is calculated by:

$$P = (P1 + P2 + \ldots + Pk)/k. \tag{7}$$

During vertex clustering, those cells which contain vertices from more than one point cloud are marked as overlapping regions. This information is used while generating texture coordinates (Section 3.1.5).

3.1.5. Texture Coordinate Generation

Texture coordinates are generated for each vertex in the simplified cloud. The vertices in the overlapping regions will have texture coordinates for two textures, and all other vertices will only have a texture coordinate for a single texture. Texture coordinates are calculated linearly as follows:

$$\begin{aligned} tex_u &= (x - x_{min})/(x_{max} - x_{min}) \\ tex_v &= (y - y_{min})/(y_{max} - y_{min}) \end{aligned} \tag{8}$$

where x and y is the vertex's 2D location, while x_{min}, x_{max}, y_{min}, and y_{max} are the minimum and maximum x-coordinate and y-coordinate of the vertices, respectively (see Figure 4).

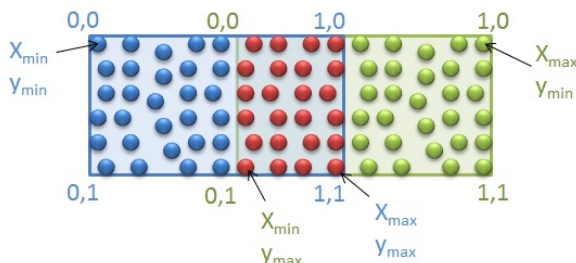

Figure 4: *Texture Coordinate Generation*

3.1.6. Triangulation

Finally the point cloud is converted to a surface. There is a lack of implementations of triangulation algorithms which can handle 3D point clouds. Those which were found which claim they do, are actually used to create terrain surfaces, and thus only do z-value approximation (2.5D triangulation). One such implementation is the one implemented by [BM05], which is the one we utilised in our implementation. The z-value approximation as implemented in this library was not good enough for our purpose, as it resulted in triangles being created across the object, instead of resulting in a decent continuous surface. Our solution included unwrapping the point cloud onto a plane, such that it resembles a terrain, the main purpose the library had been intended for. The library is then used to triangulate the unwrapped cloud, and afterwards it is re-wrapped to its original shape, along with the generated triangles.

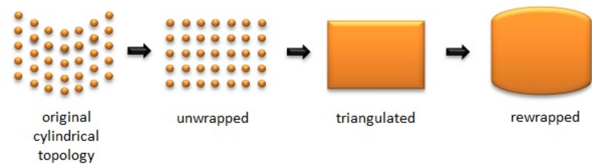

Figure 5: *An example of the triangulation process for a cylindrical topology*

3.1.7. Representation

Finally, the processed information is exported as a single .objv file. We wanted to create a file format to specifically suit our needs, yet ideally the file would be able to be opened via other programs. Therefore we chose to create a variant to the standard .obj file format. This includes the vertex and face information, texture coordinates, the camera position and direction of each viewpoint, as well as the translation and rotation matrices used during registration.

3.2. Texture and Reflectance Behaviour

For each viewpoint, apart from capturing the depth map, we also capture around 40 images (as suggested in the original PTM paper [MGW01]) using the Kinect RGB camera with the object lit from varying light directions. The captured data is then processed through the pipeline described below.

3.2.1. Highlight Detection

Since our method uses HRTI to estimate the light direction vector, first the highlights on the spheres have to be detected. The highlight on every sphere is detected by calculating the luminosity of every pixel in the area of the sphere. Bright pixels are considered to be candidates, and in cases where there are multiple candidates, the highlight is taken to be the candidate pixel closest the centre of the sphere, since according to [Gar09], fake highlights caused by inter-reflection are usually away from the centre.

The brightness calculation is based on the contribution of every component to the perceived luminance. The contribution values used are those defined in Rec. 709 [Int90], which standardizes the format of high-definition television:

$$L = 0.2126R + 0.7152G + 0.0722B. \tag{9}$$

© The Eurographics Association 2013.

3.2.2. Light Direction Extrapolation

From the position of the highlights, H_x and H_y, the center of the sphere, C_x and C_y, and the radius of the sphere, r, the light direction vector is calculated using the method proposed by Mudge et al. [MMSL06].

First, the highlight location is normalised:

$$S_x = (H_x - C_x)/r \\ S_y = (H_y - C_y)/r. \tag{10}$$

The inclination of the surface normal is given by:

$$\phi = cos^{-1}(\sqrt{1 - S_x{}^2 - S_y{}^2}). \tag{11}$$

The inclination of the light source is twice the inclination of the normal thus:

$$\phi_L = 2cos^{-1}(\sqrt{1 - S_x{}^2 - S_y{}^2}). \tag{12}$$

The other (azimuthal) angle of the light source is calculated:

$$\theta_L = sin^{-1}(S_y/sin(\phi)). \tag{13}$$

Finally, the normalised light direction vector is given by:

$$L_x = sin(\phi_L)cos(\theta_L) \\ L_y = sin(\phi_L)sin(\theta_L). \tag{14}$$

3.2.3. Image Filtering

All of the captured images contain the object as well as the three spheres and other background information. Since the light direction vectors have already been extrapolated, this phase processes each image to retain only the object part of the image. Since for each viewpoint we captured a point cloud, which has been processed and filtered, using a mapping operation the system can calculate which pixels are to be cropped.

3.2.4. Image Upsampling

Since the captured images have a resolution of 640x480 pixels, and moreover they are cropped to retain only the object portion, the resulting images would have a very low resolution. Therefore we introduced an additional phase to the pipeline, in which the cropped images are upsampled.

Our implementation makes use of JavaCV, a Java interface to OpenCV, and attempts to upsample the images to a resolution of 640x480 by using bilinear interpolation. We believe that this phase will probably not be needed when the new Kinect sensor will be released, as the resolution will be high enough to give satisfactory results even after cropping.

3.2.5. PTM Generation

Finally, a PTM for each viewpoint is generated, using the method described in [MGW01]. Our system uses the LRGB PTM format, which stores 6 coefficients for every pixel. These coefficients determine the pixel's luminosity for a particular light direction. First, for $N+1$ images, and light direction given by l_{uk}, l_{vk} for the kth image, we build matrix M, as follows:

$$M = \begin{pmatrix} l_{u0}^2 & l_{v0}^2 & l_{u0}l_{v0} & l_{u0} & l_{v0} & 1 \\ l_{u1}^2 & l_{v1}^2 & l_{u1}l_{v1} & l_{u1} & l_{v1} & 1 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ l_{uN}^2 & l_{vN}^2 & l_{uN}l_{vN} & l_{uN} & l_{vN} & 1 \end{pmatrix}$$

Matrix M is then factorised using SVD, into U, S and V.

$$(U,S,V) = SVD(M). \tag{15}$$

The light pseudo inverse matrix P is created as follows:

$$P = VS'U^T. \tag{16}$$

where matrix S' is equal to S, except for the principal diagonal, which is defined as follows:

$$S'_{(x,x)} = \frac{1}{S_{(x,x)}} \; where \; S_{(x,x)} != 0. \tag{17}$$

Since the LRGB format is used, the luminance and chromaticity components need to be separate for each pixel. The original PTM paper [MGW01] does not indicate how this is to be done. We follow work by Ranade et al. [RSK08] in implementing a method in which the luminance is simply the average of the RGB components, and the chromaticity is a weighted average, where a higher weight is given to pixels with moderate intensity, and lower weight is given to bright/dark pixels.

Multiplying matrix P by matrix L, an (N+1 by 1) matrix containing the luminance values, for each pixel, yields a matrix, C, containing the 6 coefficients for that pixel:

$$C = PL. \tag{18}$$

In order to efficiently store the coefficients as 1 byte, each coefficient is scaled and biased. 6 float scale values (λ) and 6 integer bias values (ω) are calculated, one for each coefficient, as follows:

$$\omega = -(255 * min)/(max - min) \\ \lambda = (max - min)/255 \tag{19}$$

where min and max are the minimum and maximum value of the coefficients.

Each coefficient is then scaled and biased:

$$C'_i = \frac{C_i}{\lambda} + \omega. \tag{20}$$

© The Eurographics Association 2013.

The LRGB PTM is then created in line with the file format specification [MG01].

4. Rendering

The Rendering stage involves mapping the 3D geometry information and the PTMs, and rendering the result in a viewer which would allow the user to move the camera, rotate the object as well as change light direction. Our work includes vertex and fragment shaders such that the processing is done almost entirely on the GPU.

4.1. Determining Light Direction

The user of our system is provided with a light panel to allow movement of light direction. The bounding area of the light panel represents the visible range of the object from the current camera viewpoint. Thus when the user moves the light position in the light panel, our system calculates the position of the closest corresponding vertex (v_1) in the object.

The system then calculates the light direction that the PTMs needs to be lit from. If vertex v_1 is within an overlapping region, the process needs to be repeated for each of the two contributing PTMs. The algorithm implemented to determine the light direction is the same algorithm used for HRTI (section 3.2.2).

4.2. Vertex Shader

For each vertex, the vertex shader requires the following inputs:

- $min_{x0}, min_{y0}, ..., min_{xN}, min_{yN}$, the point where each overlapping region starts (uniform)
- $max_{x0}, max_{y0}, ..., max_{xN}, max_{yN}$, the point where each overlapping region ends (uniform)
- tu_0, tv_0, tu_1, tv_1, the texture coordinate for the PTM, or at most 2 texture coordinates in cases where the vertex lies in an overlapping region (attributes)

If the vertex lies in an overlapping region, there are two PTMs which yield information about it. The vertex shader calculates the weighted contribution of the leftmost PTM (r_{left}), which depends on the position of the vertex in the overlapping region.

$$\begin{aligned} \delta &= max_{xi} - min_{xi} \\ \gamma &= tu_0 - min_{xi} \\ r_{left} &= \frac{\gamma}{\delta} \end{aligned} \tag{21}$$

where i is the number of the overlapping region.

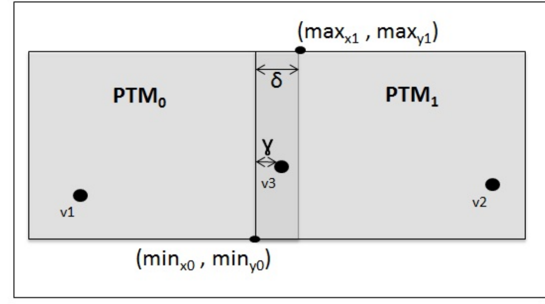

Figure 6: *The inputs of the vertex shader. Since v1 and v2 are vertices in the non-overlapping region, they will only have a single texture coordinate. v3 on the other hand, is found in the overlapping region, therefore the contribution of each PTM is calculated*

The outputs of the vertex shader are:

- tu_0, tv_0, tu_1, tv_1, the texture coordinate for the PTM, or at most 2 texture coordinates in cases where the vertex lies in an overlapping region (varying)
- r_{left}, the contribution of the leftmost PTM, if vertex lies in overlapping region (varying)

4.3. Fragment Shader

Apart from the outputs of the vertex shader, the fragment shader requires the following inputs:

- $L_0...L_N$, the light direction with which to render each of $N+1$ PTMs (uniform)
- $S_{0,0} - S_{0,5}, ..., S_{N,0} - S_{N,5}$, the 6 scale values for each of $N+1$ PTMs (uniform)
- $B_{0,0} - B_{0,5}, ..., B_{N,0} - B_{N,5}$, the 6 bias values for each of $N+1$ PTMs (uniform)
- $a_{0,0} - a_{0,5}, ..., a_{N,0} - a_{N,5}$, the 6 coefficients for every pixel for each of $N+1$ PTMs (uniform)
- $R_0, G_0, B_0, ..., R_N, G_N, B_N$, the RGB chromaticity values for every pixel for each of $N+1$ PTMs (uniform)

In the fragment shader, for each involved PTM, the required coefficients are restored to their original value (since they have been scaled and biased for compression):

$$C_i = \lambda(C_i' - \omega). \tag{22}$$

Then, depending on the light direction, and the coefficients, the luminosity Lum for each pixel is calculated as follows:

$$Lum = C_0 L_u^2 + C_1 L_v^2 + C_2 L_u L_v + C_3 L_u + C_4 L_v + C_5. \tag{23}$$

If the fragment lies in an overlapping region, the luminosity is taken to be:

$$Lum = (r_{left} Lum_0) + (1 - r_{left}) Lum_1. \tag{24}$$

© The Eurographics Association 2013.

where Lum_0 and Lum_1 are the luminosity values of the two PTMs in the overlapping region. This blending method was proposed as a way to blend medical light microscopy images [RLE*05].

Finally, we calculate the fragment's RGB colour:

$$\begin{aligned} R_{final} &= R * Lum \\ G_{final} &= G * Lum \\ B_{final} &= B * Lum. \end{aligned} \tag{25}$$

5. Results

We captured data of prehistoric artifacts at the National Museum of Archaeology, Malta. Figure 7 shows the setup used to capture a stone of an approximate width of 1.2m. Such an artifact would be difficult to translocate, therefore having a method that allows on-site capturing was important.

Figure 7: *The setup used for capturing*

As the three reflective spheres, we used ordinary billiard balls. We made sure that all three spheres were clearly visible from every viewpoint. Since the artifact was located right in front of a wall, it was only possible to capture data from two viewpoints. The 3D geometry was then processed using our method.

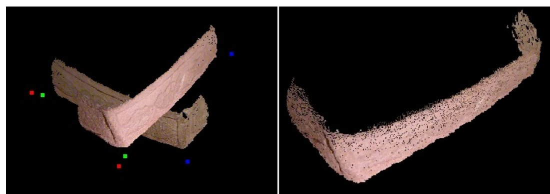

Figure 8: *Point Clouds before (left) and after (right) registration*

The images with different light directions were processed to create the PTMs. Figure 10 shows our PTMs in the c-h-i viewer, viewed as 2D light-adjustable image files [Cul10].

Figure 11 shows some screenshots of our viewer, rendering the PTMs mapped onto the processed geometry. The user is able to move the camera, rotate the object, as well as change the light direction from the light panel.

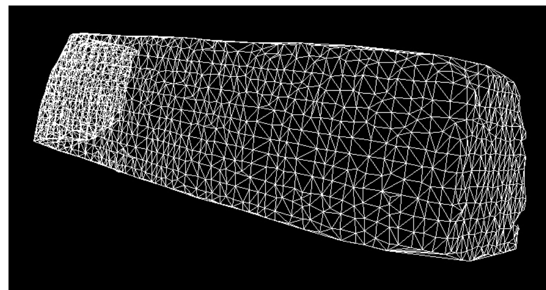

Figure 9: *The simplified point cloud is converted into a surface*

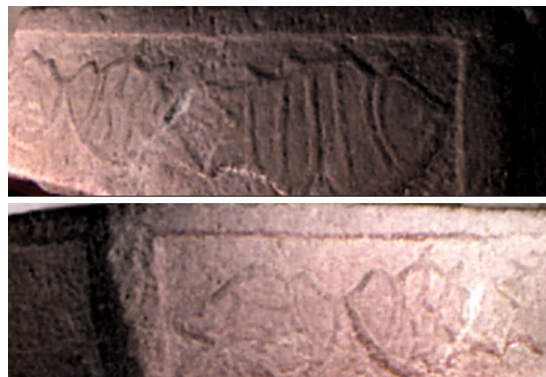

Figure 10: *PTM captured from right side of object (top) and PTM captured from left side of object (bottom)*

The whole process of capturing and processing the prehistoric stone took around 2 hours. Considering the facts that the stone could not be moved and the museum's ambient light could not be switched off, as well as the relatively low resolution that Kinect's camera provides, the results are quite satisfying.

We also captured a Bronze Age artifact the width of which is around 0.25m. One of the resultant PTMs is shown in Figure 12. This time results were less satisfactory, the reason being the quality of the PTMs. Since the Kinect sensor is only capable of providing depth data for objects which are at least 80cm away from it, the object is occupying only a small portion of the 640 by 480 image.

6. Conclusions and Future Work

In this work we have proposed a method with which the geometry, texture, and reflectance behaviour of a real-world object can be captured. Our method requires only a Kinect Sensor, 3 reflective spheres, and a light source, and can be used on-site. Our work also extends the use of Polynomial Texture Maps to three dimensions. By capturing overlapping geometry data and PTMs from different viewpoints, the ge-

© The Eurographics Association 2013.

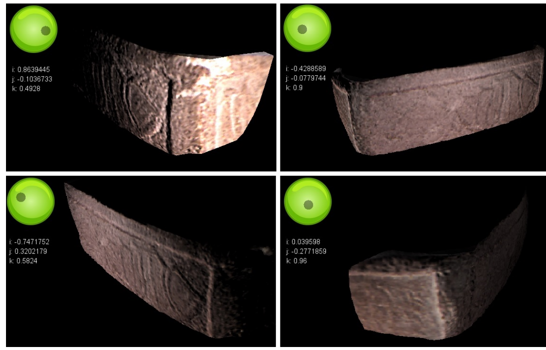

Figure 11: *Object being viewed from different viewpoints and with different light directions*

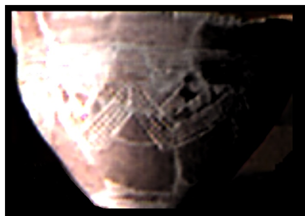

Figure 12: *PTM showing an artifact from Bronze Age*

ometry and PTMs can be mapped to enable realistic and interactive examination of the object. Acquiring data of a real-world object and processing it only takes around 2 hours, and does not require any special skills to be carried out.

While our proposed pipeline supports 360° coverage of real-world objects, our current implementation only supports processing and rendering of data captured from two viewpoints. We hope that in the future the implementation will be extended as we believe it would demonstrate our method better. Although the current Kinect technology has certain limitations which have affected our results, future improvements in this technology will render our work even more useful and much better results will be evident. Also, more work can be done on the representation of the geometry and especially PTM files. Using our current method the geometry data file would be about 150Kb and each PTM would take up around 2.5Mb of space. Although this looks reasonable, if Kinect technology is improved and offers a higher resolution, PTM compression methods will be useful such that the available resolution is utilised in full and yet file sizes are kept to a minimum.

7. Acknowledgements

We would like to thank the staff of Heritage Malta and the National Museum of Archaeology for their support. Assisting us in acquiring data of prehistoric artifacts made possible evaluation of our work.

References

[BM05] BEN-MOSCHE B.: Delaunay triangulation. `http://www.cs.bgu.ac.il/~benmoshe/DT/`, Last Accessed: 11/07/2013, 2005. 4

[Cul10] CULTURAL HERITAGE IMAGING: Reflectance transformation imaging, 2010. 2, 7

[DCCS06] DELLEPIANE M., CORSINI M., CALLIERI M., SCOPIGNO R.: High quality ptm acquisition: Reflection transformation imaging for large objects. In *The 7th International Symposium on VAST* (2006), Eurographics Association. 2

[Gar09] GARCIA J. C.: Rti-based techniques and tools for digital surrogates, 2009. 4

[HOMG03] HEL-OR Y., MALZBENDER T., GELB D.: Synthesis and rendering of 3d textures. In *Proceedings of Texture 2003 - 3rd International Workshop on Texture Analysis and Synthesis, Nice, France* (2003). 2

[IKH*11] IZADI S., KIM D., HILLIGES O., MOLYNEAUX D., NEWCOMBE R., KOHLI P., SHOTTON J., HODGES S., FREEMAN D., DAVISON A., FITZGIBBON A.: Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology* (2011), pp. 559–568. 2

[Int90] INTERNATIONAL TELECOMMUNICATION UNION: Itu-r recommendation bt.709, 1990. 4

[MG01] MALZBENDER T., GELB D.: Polynomial texture map (.ptm) file format, 2001. 6

[MG02] MALZBENDER T., GELB D.: Imaging fossils using reflectance transformation and interactive manipulation of virtual light sources. In *Palaeontologia Electronica* (2002). 2

[MGW01] MALZBENDER T., GELB D., WOLTERS H.: Polynomial texture maps. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), SIGGRAPH '01. 2, 4, 5

[MMK03] MESETH J., MÜLLER G., KLEIN R.: Preserving realism in real-time rendering of bidirectional texture functions. In *OpenSG Symposium 2003* (2003), Eurographics Association, Switzerland, pp. 89–96. 2

[MMSL06] MUDGE M., MALZBENDER T., SCHROER C., LUM M.: New reflection transformation imaging methods for rock art and multiple-viewpoint display. In *VAST'06* (2006), pp. 195–202. 2, 5

[MVSL05] MUDGE M., VOUTAZ J.-P., SCHROER C., LUM M.: Reflection transformation imaging and virtual representations of coins from the hospice of the grand st. bernard. Eurographics Association. 2

[RLE*05] RANKOV V., LOCKE R. J., EDENS R. J., BARBER P. R., VOJNOVIC B.: An algorithm for image stitching and blending. In *Proceedings of SPIE* (2005), pp. 190–199. 7

[RN10] RAJIV P., NAMBOODIRI A. M.: Image based ptm synthesis for realistic rendering of low resolution 3d models. In *Proceedings of the Seventh Indian Conference on Computer Vision, Graphics and Image Processing* (2010), ICVGIP '10, pp. 345–352. 2

[RSK08] RANADE A., SHANKAR S., KASHYAP S.: *Image Relighting using Polynomial Texture Maps*, 2008. 5

[TFS*12] TENEDORIO D., FECHO M., SCHWARTZHAUPT J., PARDRIDGE R., LUE J., SCHULZE J. P.: Capturing geometry in real-time using a tracked microsoft kinect, 2012. 2

© The Eurographics Association 2013.