Abstract—We present a new 3D reconstruction pipeline for digital preservation of natural and cultural assets. This application requires high quality results, making time and space constraints less important than the achievable precision. Besides the high quality models generated, our work allows an overview of the entire reconstruction process, from range image acquisition to texture generation. Several contributions are shown, which improve the overall quality of the obtained 3D models. We also identify and discuss many practical problems found during the pipeline implementation. Our objective is to help future works of other researchers facing the challenge of creating accurate 3D models of real objects.

Keywords—digital preservation; 3D scanning; range images

I. INTRODUCTION

Digital reconstruction of 3D models from range and color images is a very active research field, but still includes many challenges. In this context, there are two main surveys [1], [2] presenting an entire reconstruction pipeline. Also, pioneer works focusing on digital preservation of cultural heritage are presented in [3], [4], [5]. These works highlight the difficulties to be overcome in scanning complex objects.

In this paper, we show how we built a functional 3D reconstruction pipeline, aiming for high quality results required in the digital preservation of natural and cultural assets. Our contributions and the rationale behind the choices we make are meant to help future works in this area [6], [7].

In our work we used a commercial 3D scanner, the Vivid 910 from Konica Minolta. However, the reconstruction software shipped with the scanner has several limitations: the alignment of views is an arduous process; the mesh integration sometimes generates incorrect surfaces that need to be manually corrected; the hole filling does not work in holes with complex topologies; and the generated texture is of low quality and incorrectly parametrized in all but the simplest objects. All these factors make inappropriate its use for digital preservation, and motivated our development of a working high-quality 3D reconstruction pipeline.

Several steps compose a complete 3D reconstruction pipeline, as shown in Fig. 1.

First, the data is acquired from different and sufficient viewpoints. Next, the data is aligned into a common reference frame in a process known as registration. After alignment, follows the mesh integration stage where data from all acquired views are combined. Eventual holes due to incomplete data acquisition are usually filled after the integration step. Then, a 3D model with its textures (i.e. diffuse color, specular and normal maps) is generated. Finally, mesh simplification may be performed to improve rendering performance and storage costs. We follow this sequence of stages to present our solutions as well as other related works.

II. THE 3D RECONSTRUCTION PIPELINE

A. Acquisition

There are several types of acquisition devices to gather depth information from an object: laser scanning, multi-view stereo, shape from structured light, shape from silhouette, contact digitizers, among others. From all these techniques, laser scanning is the most precise [8], being our choice.

One current avenue of research is dedicated to improving the quality of the acquired data. Nehab et al. [9] combine depth information from a triangulation scanner with normal information from photometric stereo. Park and Kak [10] proposed a technique to capture optically challenging objects through the usual laser triangulation technique with some modifications. Any improvement in the quality of acquired data is very welcome since it surely improves the fidelity of the 3D reconstruction.

The first obstacle we faced was how to separate the object from the background. The segmentation by color is problematic, because it needs a controlled acquisition
background, and places restrictions on colors present on the scanned object. Scanning the object placed over a black surface (to avoid laser capture), is not a good option either, because dark regions of the object need higher laser power; therefore leading to the capture of the black surfaces used as support for the object.

Our contribution to this stage is a new method able to separate the object from the background [11]. We assume that the object is always scanned over a support plane (e.g., a table or the floor). This is not a severe restriction: when scanning large objects, usually there is no background to eliminate; and with small objects, they need to be placed over a support anyway, that will inevitably be captured too.

Our approach automatically detect and remove the acquired support plane, since we know that there is no information below the plane, and the object data is above it. If we project all captured 3D points $x$ into the support plane normal direction $n$ (using a dot product $d = x \cdot n$), the points on the plane would project at the same arbitrary value $d$, the points on the object would be spread with values larger than $d$, and there would be very few values below $d$ (due to noise). If we build a histogram of the distribution of these values, one can find a distinct profile (see Fig. 2). Note that all those operations happen on the local coordinate system used by the scanner.

Our method tries different normal directions (sorted by occurrence on the data points), searching for a histogram that matches the profile of Fig. 2. When one is found, we select the points on the peak of the histogram, and refine the plane parameters with MSAC [12]. The refined plane is accepted and found if it conforms to a plane thickness threshold (usually around 2 mm), minimum plane percentage of total points (usually around 5%), maximum below plane percentage of total points (usually around 2%) and minimum above plane percentage of total points (usually around 10%). If any of these conditions fail, a new plane normal direction is tried. The thresholds were obtained empirically, and can be adjusted to different compromises between precision and detection. Once the plane is obtained, only the points above the plane are marked as belonging to the object.

One advantage of our method is that control of the background colors is unnecessary, what is useful for acquisitions made outside a controlled environment. Besides, our method does not place any restrictions on scanning parameters (e.g., focus distance or laser power). Finally, the detected plane can be helpful in the mesh integration stage, because it defines a half-space that is known to be outside the scanned object. Fig. 3 shows the automatic support plane detection.

![Range image of a real insect (beetle), showing a plane difficult to detect: (a) original range image, with several disconnected patches belonging to the support plane; (b) result from our automatic support plane detection and removal algorithm.]

**Figure 3.** Range image of a real insect (beetle), showing a plane difficult to detect: (a) original range image, with several disconnected patches belonging to the support plane; (b) result from our automatic support plane detection and removal algorithm.

### B. Registration

The objective of this stage is to find a $4 \times 4$ transformation matrix for each captured view to achieve the alignment into a common object coordinate system. Rusinkiewicz and Levoy [13] and Salvi et al. [14] present several algorithms that can be used in the registration stage.

In our pipeline, we use a pairwise ICP alignment [13], followed by a global registration step using Pulli’s algorithm [15]. For each pair of neighboring views with sufficient overlap, we find the transformation matrix that aligns the second view with the first, using a modified version of the ICP algorithm, presented below. Currently, we manually pre-align the views; however, automatic pre-alignment techniques like in [16] can be used to improve this task.

Our contribution regarding this stage is a new two-phase ICP algorithm. We needed an algorithm with good convergence properties (to reach the correct alignment), and with maximum precision. To achieve this, the first phase uses an ICP variant with the point-to-plane error metric, a closest-compatible approach for pair generation, normal space sampling with random selection of points on both views, and rejection of the farthest pairs [13]. This promotes excellent convergence, but with limited precision.

When this first phase converges, we move on to the second phase of the ICP algorithm. The changes from the first phase are: we use all points on both views (instead of random sampling); we add a maximum pair distance into the pair compatibility test, which is usually very small (e.g., around 0.7 mm), instead of rejecting the farthest pairs; and use a

![Example of a distribution of range points over a candidate support plane normal direction.](Image)

**Figure 2.** Example of a distribution of range points over a candidate support plane normal direction.
slightly more restrictive normal compatibility test for pairs. We still use the point-to-plane error metric during error minimization. This version of ICP has limited convergence, but excellent precision. As the first phase already reached an almost optimal alignment, the second phase just improves the precision of the result. As the pair compatibility test is very restrictive, we achieve good outlier elimination, which is essential for a precise alignment.

Fig. 4 shows an experimental result from this two-step ICP-based approach; even with the low overlap and bad initial position, the result converged to a precise alignment.

After the pairwise alignments, the global registration algorithm of Pulli [15] improves the final alignment, spreading the errors equally between all view pairs.

![Figure 4](image)

Figure 4. Our two-phase ICP-based approach: (a) initial position of two views, far from being aligned; (b) result from the first phase of our algorithm; (c) final result, with alignment precision enhanced by the second phase of our algorithm. Red is used to represent the matching pairs.

C. Mesh Integration

After the registration, we have several overlapping partial meshes, one for each captured view. The next stage of the reconstruction pipeline must integrate them to build a single triangle mesh for the object. There are several approaches for mesh integration [1]: Delaunay-based methods, surface-based methods, parametric surfaces and volumetric methods, all of them presenting limitations.

Delaunay-based methods use the Delaunay complex \( D(S) \) associated to a set of points \( S \) in \( \mathbb{R}^3 \). This complex imposes a connectivity structure to the points, and the methods extract a sub-complex of \( D(S) \) to represent the integrated surface. We can highlight the techniques that use power crusts [17]; cocones [18]; and eigencrusts [19]. The limitation of such algorithms is that they are sensitive to noise and outliers because they interpolate the data points. The solution for this requires a preprocessing step capable of “cleaning” the input data (e.g. [20]). Also, these algorithms are extremely costly in performance, what usually limits the data set size that can be processed.

Surface-based methods create or manipulate surfaces directly. In such cases, each range view defines a partial surface of the object, built from the triangulation of neighbor points in the captured grid of the range view. Examples of such methods are zippered meshes [21] and ball-pivoting [22]. Some of these algorithms can catastrophically fail in regions of high curvature, as shown in [23]. Topologically incorrect solutions can also occur due to outliers in the input views. Other problem is that in general, these methods use fragments from the 3D models of each view. Without further post-processing, this generates noisy surfaces because each view fragment is noisy.

Parametric surfaces methods can deform an initial approximation of the object through the use of external forces and internal reactions and constraints, as in [24], [25]. Other approaches use one or more analytically generated surfaces to represent the integrated model, like Radial Basis Functions [26], Partition of Unity [27] and Poisson Surface Reconstruction [28]. Most of these approaches use concepts from volumetric methods (e.g. implicit functions). One limitation of parametric surfaces is the difficulty in representing sharp corners, as most methods assume a continuous differentiable function to represent the surface. Another related problem is finding a balance between over smoothing effect and non-elimination of noisy surfaces. They may also fill holes incorrectly.

Volumetric methods use an implicit volumetric representation of the final model. So, each voxel has a value corresponding to the distance between the voxel and the integrated surface. The object surface is obtained through isosurface extraction by some variant of the Marching Cubes algorithm [29]. Unlike the parametric surfaces approach, the volumetric methods do not attempt to calculate the distance function analytically; it is defined solely through the interpolation of the samples of each voxel. We can highlight VRIP (Volumetric Range Image Processing) [23]; Consensus Surfaces [30], [31], [32]; Marching Intersections [33]; and the method with unsigned distance fields of Hornung and Kobbelt [34]. Volumetric methods have the advantage of using all available information (which helps to attenuate the noise of the input data) and ensuring the generation of manifold topologies [23]. Their main limitation is their performance, both in terms of memory usage and processing time. To achieve high fidelity, the voxel size should be approximately equal to the scanner sample distance, which is usually small (around 0.33 mm). Small voxels generate large volumes; therefore great effort is necessary to make the processing of these large data sets viable.

Choosing an approach for mesh integration is difficult, due to the large number of possibilities. We chose volumetric methods because they impose fewer restrictions to reconstructed objects; have relatively easy implementations; offer an easy way to change the precision of the output (by varying the voxel size); can easily support the space
carving technique [23] to help outlier elimination; and can work in the presence of low quality input data. Besides, Kazhdan et al. [28] compared several recent algorithms, and VRIP, despite being old, still produces better results than several recent techniques. Even compared with the Poisson Surface Reconstruction method proposed by Kazhdan, VRIP still has some advantages, like better hole filling due to the use of scanner line-of-sight information.

We implemented, tested and modified three integration algorithms: VRIP [23], used in “The Digital Michelangelo Project” [4]; Consensus Surfaces [30], used in “The Great Buddha Project” [5]; and our new algorithm, developed to solve the limitations present in the two previous algorithms. VRIP in general achieves good results, but presents some artifacts near corners and thin surfaces. Consensus Surfaces, even with some current improvements [31], [32], still generates incorrect results in regions near occlusions, as these regions rarely can achieve consensus.

To solve these drawbacks, we developed a new algorithm that combines elements from both VRIP and Consensus Surfaces. Our new algorithm is based on two phases. In the first one, we use a slightly modified version of VRIP, together with a space carving method, to generate an initial volumetric representation. Our modification on VRIP is a new weight curve (see Fig. 5), that gives more weight to outside voxels than to the ones inside the objects. This attenuates the artifacts of VRIP in corners and thin surfaces, at the cost of a small misplacement of the surfaces in the first phase. The rationale behind this new weight curve is simple: voxels inside the object are not visible, so their distance is incorrectly inferred near corners and thin surfaces; therefore, reducing their weight attenuates these artifacts. Our space carving takes into consideration only the object data, and optionally the support planes detected in the acquisition stage, having as main goal the outlier elimination. The volumetric result of this first phase works as a consensual basis for the second phase of the algorithm.

The second phase builds the definitive volumetric representation, integrating only measurements in consensus with the result obtained in the first phase. The consensus is tested at each candidate voxel, between the normal on the closest surface point of each view and the gradient of the volumetric result from the first phase. The space carving performed on the first phase is also used to eliminate outliers, here standing for the incorrect data outside the object. We must note that we use line-of-sight signed distances on the first phase for performance, and Euclidian distances on the second phase for precision and correction of the hole filling later on. Another consideration is that the small misplacement of the surfaces in the first phase cause no harm, because we are interested primarily in the normal vector field built in the first phase, which is not affected by the misplacement.

With our algorithm, we eliminate the artifacts of VRIP near corners and thin surfaces, and generate good results near occluded regions. Fig. 6 shows a comparison of results from the integration algorithms discussed previously.

Figure 5. Distance weight curves for VRIP. The original curve [23] is shown with the dashed line, and our new curve with the solid one. Our new curve is a simple concatenation of two bezier segments, and reduces the artifacts near corners and thin surfaces. This factor ranges from 0.0 to 1.0 (according to the signed distance), and is multiplied by the other weight factors used in VRIP. Negative values of distance are outside the object.

D. Hole Filling

The acquisition process is usually incomplete. Deep recesses and occlusions prevent the capture of some parts of the objects. This requires some efforts to complete the captured data to allow the generation of a “watertight” model, necessary for several applications such as user visualization and creation of replicas.

Some integration algorithms fill holes automatically, like the ones based on parametric surfaces [26], [27], [28]; however, the results are not always topologically correct. Some simple techniques catastrophically fail in holes with complex topology, common in real objects, when they assume that the holes have a disc topology.
In our pipeline, we chose the volumetric diffusion algorithm by Davis et al. [35], because it can handle complex topologies satisfactorily. Besides, it is a volumetric technique that works well with our mesh integration stage. The idea of the algorithm is to diffuse the values on observed voxels into voxels without data, similar to a 3D blurring operation. Space-carving information, although not necessary, can help the algorithm produce a more faithful reconstruction.

The volumetric diffusion algorithm suffered some criticism by Sagawa and Ikeuchi [32], but we disagree with their assessment. In our experiments, the Volumetric Diffusion generated excellent results, mainly due to the quality of our integration method. The explanation lies on the characteristics of the integration algorithms used. Sagawa and Ikeuchi [32] use the Consensus Surfaces, which usually generates incorrect results near holes. So, when propagating these incorrect information to fill holes, bad results should be expected. Since our new integration method eliminates incorrect data near holes, and space carving data from the first integration phase is available, Davis’ method is able to generate good results in these challenging cases. Therefore, we can say that Volumetric Diffusion is a good technique, but depends on a good mesh integration to work successfully. Fig. 6(d) shows Davis’ method result after our integration algorithm was performed.

E. Mesh Generation

We use the well established Marching Cubes algorithm [29] to generate a triangle mesh from the volumetric representation of the previous stages. We use the disambiguation method of Chernyaev [36] to ensure the generation of manifold topologies.

The only drawback of this approach is the generation of very thin triangles (a.k.a. slivers) in some parts of the generated model. A mesh simplification technique like [37] can eliminate these triangles, resulting in a more homogeneous mesh, useful for the next stages of the pipeline.

F. Texture Parametrization

The mesh generation concluded the geometric part of the reconstruction problem. However, we still needed to calculate the surface properties (i.e. color and specularity). These properties are usually represented by textures. Therefore, we need to be able to apply textures to the generated model.

The goal of this stage is to generate 2D texture coordinates \((u, v)\) for each 3D vertex of the model, which map a position into the texture image. This process can be seen as “skinning” the model through cuts and planifications.

There are several methods to perform texture parametrization [38]. For our purposes, we needed a parametrization that minimized distortion, being at the same time as homogeneous as possible. In our implementation, we used a simple texture atlas approach, for speed and easiness of implementation. We segmented the model into almost planar regions, starting from a random seed triangle and growing the region while the normals of the faces are within a threshold (usually 30°, to prevent the generation of too many small regions) from the average normal of the region. Each region is then planified; this is done by calculating the principal axis of the vertices in question [39]. The axis closest to the average normal of the region is then used as the normal of the plane, and the other two axes define the \(u\) and \(v\) directions in the texture space. The result is a 2D projection (in mm) of each region. After all regions are planified, a texture atlas is generated, packing all regions into a single parametrization space (see Fig. 7). As we know the size of each region in millimeters, it is easy to define the image size in pixels necessary to achieve a desired resolution in pixels per millimeter.

We must notice that any parametrization can be used with our pipeline. An extensive review of more complex parametrization alternatives is presented in [38]. For example, using a global parametrization approach as [40] should improve the parametrization quality, compared with our simple approach. Our approach was not intended to be a definitive solution; we wanted first to complete our pipeline, and then improve its stages according to the time available and impact on the final precision of the results. Unfortunately, we were not able to experiment with other texture parametrization schemes.

It is important to note that the trivial solution of generating an atlas where each triangle corresponds to a planar region is a really bad choice of texture parametrization. The performance of real time rendering suffers greatly due to lack of spatial coherence, and mip-mapping [41] becomes almost impossible to accomplish, consequently reducing the quality of the renderings. Therefore, minimizing the number of regions on the atlas is also an important criterion when choosing a good texture parametrization scheme.

G. Surface Properties Calculation

The main surface property we need to calculate is the reflectance or surface color. Additional properties, like specularity, are also useful in high fidelity reconstructions.

Acquiring accurate color information from the object is more challenging than it appears. Usually, we do not have complete control over the incident illumination on the scanned object. Even when this is tried, the simple illumination models commonly used in Computer Graphics (e.g. Phong, Blinn, Torrance-Sparrow [41]) are not physically realistic, since they ignore indirect illumination and object inter-reflections. Bernardini and Rushmeier [1] present the main techniques used to estimate the surface properties, including the compensation of the illumination parameters.

Another practical difficulty is that the commercial 3D scanners available usually return color information in low resolution. For example, the Vivid 910 we used returned images with 640 × 480 pixels of resolution, and the color
is not reliable. This leads to the use of a different high resolution camera to acquire color images, and the need of calibration between this camera and the scanner data, another source of imprecision on the final result.

There are two approaches to generate the surface properties. We can either generate color and illumination per vertex, as the models are usually high-poly; or we can generate them directly into the texture space, using the parametrization of the previous pipeline stage. The former is used when only data from the 3D scanner is available, while the latter is needed when using high resolution cameras.

Our current implementation still does not calculate an accurate photometric modeling. We use the simple vertex color approach, and we have been improving it using high resolution cameras [42]. Specularity is not yet estimated, too. To generate the vertex colors, we calculate a weighted average of the colors on all views that observe each vertex. The weight we adopted is the angle between the scanner line-of-sight and the vertex normal. This is done because the data observed at an angle are less reliable than the data facing directly the scanner. Although simple, our method generates good results, as shown in Section III.

H. Texture Generation

Texture generation combines the results of the two previous stages of the pipeline: texture parametrization and surface properties. Our objective is to encode the surface properties into one or more images (i.e., textures). These will be used when rendering the reconstructed model (see Fig. 7).

This stage and the previous two are very dependent on the algorithm used. Sometimes, they are condensed in a single stage [43]; in other cases, they are strongly related to the acquisition devices used [44]. We prefer to separate these three stages, so that different techniques can be tested, and at the same time easily integrated in our pipeline.

As we explicitly generate the parametrization and vertex color, the texture generation is straightforward. We render the model using the texture coordinates \((u, v, 0, 0)\) as the \((x, y, z)\) coordinates of the vertices respectively, and using the calculated vertex colors. The 3D graphics card interpolates the color across each face using Gouraud shading [41]. We use an orthogonal projection matrix, and a render target of the size of the desired texture. The same rendering technique can be used to generate other textures, like a normal map (encoding each vertex normal as a RGB triplet), or a specular map if specularity information is available (encoding each vertex specular color and exponent as a RGBA tuple).

A question that might arise is why the need to generate textures by Gouraud vertex interpolation, instead of directly rendering the object using the available vertex colors. The answer is in the next pipeline stage, mesh simplification. As our model is usually at excessively high polygon counts, the generated texture has good resolution. After the mesh is simplified to a more manageable polygon count, the surface properties are still preserved on the generated textures (i.e., diffuse color, normal map and optionally specular map).

We found another practical problem when automatically generating textures: the perimeters of each parametrized region usually causes problems with mip-mapping [41]. This occurs due to the bilinear interpolation made when accessing texture maps. This appears as “cracks” on the final model that highlights the boundaries of the segmented regions. To solve this, we expanded the colors from the regions into the unused texture spaces, using a diffusion technique. We created this technique inspired on the Volumetric Diffusion [35] used for hole-filling, but here the diffusion is 2D and we propagate color instead of distance values. This works like a blurring filter, but only affecting the unused pixels of the texture, and using only the colors propagated from the regions. For each unused texel adjacent to used ones, the unused texel receives the average color of their valid neighbors. This way, each iteration through the entire texture enlarges the valid regions by 1 pixel. This process is repeated until all unused texels receive average colors from their neighbors. This technique can be used on any chart-based parametrization scheme, and is usually effective to solve the “crack” problem. Fig. 8 shows the problem and the solution with our method, while Fig. 7 shows an example of textures with diffused regions.

I. Mesh Simplification

An optional pipeline stage consists of reducing the triangle count on the model to improve its rendering performance and storage costs. After capturing the geometric, color and eventually specular properties into textures, we can perform mesh simplification and still maintain the visually high accuracy of the source model.

When dealing with digital preservation, this step is not essential, since we want precise results. However, the Marching Cubes algorithm used in the pipeline can generate much
more triangles than necessary to accurately represent the
model, mainly in almost planar regions. So, a mesh sim-
plification procedure can improve the performance keeping
high accuracy. Another important fact is that we are able to
generate a normal map for the model that helps preserve the
visual accuracy even when low-poly models are used.

There are several approaches for mesh simplification.
The technique of Garland and Heckbert [45], improved
by Hoppe [37] is fast and generates accurate results when
reducing moderately the polygon count, which is the goal of
digital preservation. It can be combined with a progressive
mesh representation [46] to allow the generation of different
levels of detail for each object.

We still did not implement mesh simplification into our
pipeline. However, we intend to perform mesh simplification
after texture generation; this way, we are able to generate
maximum quality normal maps, and the textures can guide
the mesh simplification, thus minimizing texture distortion.

III. RESULTS AND FUTURE WORKS

We used our pipeline to reconstruct several objects,
ranging from artworks to fossils. Those objects were se-
lected to “stress test” the pipeline, with complex geomet-
ries and optically uncooperative materials. Table I shows
characteristics of some reconstructed objects, presented in
Fig. 9. In general, we are able to generate good qua-
Figure 8. Example of the “cracking” problem due to the mip-mapping
of automatically generated textures: (a) rendered result with the original
texture map; (b) rendered result with colors diffused into the unused texture
space, with “cracks” eliminated.

ble, therefore temporary files, a cache mechanism, and
cache-friendly algorithms are necessary;

- Current mesh integration algorithms still generate false
  or incorrect surfaces. This directly impacts the accuracy
  of the final models, showing that further research in this
  area is still needed;

- Mathematically evaluating accuracy of the results is still
difficult. When reconstructing real objects, we do not
have a “ground truth” to compare the generated model
to. Even producing a test object from a previous 3D
model, to scan and reconstruct it later, does not solve
the problem, as any manufacturing process introduces
inaccuracies that make the real object different from
the source 3D model. This makes a simple comparison
between the source and reconstructed model incorrect.
A valid quantitative approach to compare reconstruction
methods still need to be developed;

- Color images acquired with laser triangulating 3D
scanners are usually of low resolution. This limits
the generation of accurate textures for the 3D model
without using additional high-resolution cameras.

Some future works can focus on improving several stages
of the pipeline. An interactive tool to help planning the
capture would be useful to minimize the effort during
acquisition. Using some automatic pre-alignment for each
pair of views would reduce the amount of human labor to
generate the models. Improving the quality of the alignment
would improve the precision of the resulting models. The
development of better integration algorithms is another im-
portant avenue of research, so that only precise surfaces are
generated. Using camera calibration techniques to combine
the acquired geometry with high resolution photographs of
the object would improve the quality of the textures. Texture
quality can be further enhanced using more complex texture
parametrization schemes. Much of our work focused on the
geometry reconstruction stages, and now we aim to work on
the latter stages of the reconstruction pipeline [42].

IV. CONCLUSION

The purpose of this work is to show a complete and
functional solution for the 3D reconstruction problem ap-
plied to digital preservation. There are lots of algorithms
and possibilities to build such a pipeline; we present our
particular solution, and the reasoning behind the selection
of the algorithms for each stage of the pipeline. We are able
to reconstruct complex objects with good accuracy, proving
that our approach is functional.

Our work was intended as a baseline for future research,
allowing a comparison of new techniques with a reliable
reference. Besides, a clear division of the reconstruction pro-
cess into a pipeline with independent states allows an easier
integration of alternative techniques into the reconstruction
tool.
Our main contributions are: a new support plane detector to automatically separate the object from the background in the acquired range data; a new two-phase ICP-based algorithm to achieve at the same time good convergence and good precision; a new volumetric integration algorithm that overcomes drawbacks from both VRIP and Consensus Surfaces, working nicely with the Volumetric Diffusion algorithm for hole-filling; a functional way to generate textures for the geometric models using a simple texture atlas approach combined with a per-vertex calculation of surface color; and an automatic rendering of the texture image using common 3D graphic accelerators. Another important contribution is a general overview of the entire pipeline, and how each stage interacts with the other stages; attention to this is paramount when building a functional 3D reconstruction pipeline.

We hope our work helps other researchers facing the daunting task of building a 3D reconstruction pipeline for digital preservation, facilitating its achievement.

ACKNOWLEDGMENT

The authors would like to thank CAPES, FINEP and CNPq for funding of this research.

The proposed pipeline was applied in the digital reconstruction of several objects, in particular artifacts from the Metropolitan Museum of Art in Curitiba; fossils from the Natural History Museum of UFPR; insects from a collection from the sector of Biological Sciences of UFPR; artworks present in the UFPR President’s office; and a 30-years old partial replica of the statue of the prophet Habacuc, made by Aleijadinho, present in the Museum of Fine Arts of UFMG. We would like to thank all professors and personnel that allowed our access to these sample objects to validate our experiments in real digital preservation scenarios.

REFERENCES


Figure 9. Results from our 3D reconstruction pipeline: (a) reconstruction of a beetle, challenging because the small scale of the details and dark colors; (b) reconstruction of a metal statue of a rooster, challenging because the specularity of the object material and thin gaps between the feathers; (c) reconstruction of a *protocyon* fossil (an ancient American wolf), challenging because the complex topologies, occlusions and thin surfaces; (g) reconstruction of a bronze statue created by the Brazilian artist Carybé; (h) reconstruction of a marble statue shown at the UFPR President’s office, challenging because of the bad quality of the acquired data, due to shininess and translucency of marble; (i) reconstruction of a partial 30-year old plaster replica of the statue of prophet Habacuc, created by Brazilian artist Aleijadinho; (d), (e), (f), (j), (k) and (l) color images of the real objects. In all cases, high quality reconstructions were achieved.


