Semi-Autonomous Digitization of Real-World Environments

Mikhail M. Shashkov, Connie S. Nguyen, Mario Yepez, Mauricio Hess-Flores, and Kenneth I. Joy
Institute for Data Analysis and Visualization
University of California, Davis, USA
mmshashkov@ucdavis.edu, csnguyen@ucdavis.edu, myepez@ucdavis.edu, mhessf@ucdavis.edu, kenneth.i.joy@gmail.com

***Abstract*—In recent years, the computational power and graphics capabilities of our computers and gaming consoles has advanced to the point that we can render photo-realistic real-world environments in real-time. This capability has been utilized to make immersive games, serious games, and simulations. While computers and consoles are affordable for the consumer, the technology used to scan environments can be too expensive or time-consuming for most developers. In this paper, we seek to find a viable low-cost alternative to these expensive technologies by analyzing the efficacy of cheap hardware (Kinect and personal cameras), computer vision algorithms (KinectFusion and structure-from-motion), and post-processing tools (MeshLab) in the context of a popular free game engine, Unity. Our results demonstrate workflows that are viable for the needs of many developers.**

***Keywords*—real-world environments; multi-view reconstruction; structure-from-motion; Kinect; Unity;**

I. Introduction

Recent progress in computational power and graphics capabilities has made it possible to have photo-realistic environments in real-time interactive systems like games. Many games, such as the popular racing game Gran Turismo 5®, have utilized these capabilities to use real-world environments in their games [1]. Figure 1 shows part of the Abbey of St. Galgano in Italy from both a real photograph and Gran Turismo 5® [2]. While the renders only got better in Gran Turismo 6®, the process apparently involved over 100,000 photos, laser scans of the track surface, and helicopter and satellite imagery [3]. The year long process (per track) resulted in an impressive track accuracy of plus or minus one centimeter. Is there a way to achieve comparable results in less time? We explore this question in the paper by examining tools for data acquisition and data refinement. Finally we use Unity to experience our data in a real game engine.

For data acquisition, we explore two methods. First, we analyze the Microsoft Kinect®, a popular commercially available camera and depth sensor. We will be using the KinectFusion algorithm [4]. One of the main limitations of the Kinect for our intended application is that the hardware uses infrared sensors in order to derive depth. The sun emits energy in the infrared spectrum, which intereferes with the Kinect's structured infrared light patterns. This makes digitization impossible in outdoor environments. Our second data acquisition method, structure-from-motion, depends only on color information from photographs and thus can be used for both outdoor and indoor environments. In particular, we will be using Visual SfM [5], [6] to create spare and dense reconstructions from photographs.

In computer games it is often necessary to use meshes of various complexity, especially for level of detail (LoD) systems; this is opposed to the fields of vision and photogrammetry, where every detail is critical. Thus, it is important to verify that our data can be manipulated complexity while preserving quality. The point cloud data produced by both of our acquisition methods is easily manipulated by MeshLab (http://meshlab.sourceforge.net), an open-source system for viewing, processing, and editing meshes. In particular, we utilize algorithms for initial meshing and mesh simplification whose parameters directly influence the resulting mesh quality.

Finally, we utilize our data by creating a simple game using Unity (http://unity3d.com), which is a popular and free game engine. We verify that our meshes are adequately simple by maneuvering through the environment with real-time mesh-based collision.

Ultimately, this paper shows that a combination of several low-cost technologies can be used to rapidly develop games featuring levels based on real-world environment data. We contribute a viable low-cost workflow, verify it by scanning our own environments, and provide an analysis of ease, limitations, and other considerations. This enables independent (indie) developers and scientists developing serious games to utilize real-world environments in their efforts. Related work and background on our tools and their underlying algorithms is discussed in Section II. An analysis of tools and their limitations is discussed in Section III. Conclusions and future work is discussed in Section IV.

Fig. 1. Photograph at the Abbey of St. Galgano, Italy (left). Screenshot from a course in the game Gran Turismo 5®(right).

II. Background and Related Work

We provide a general background on computer vision and contributions towards scene reconstruction in Section II-A with a particular focus on structure-from-motion. We discuss cutting-edge reconstruction algorithms that utilize Red-Green-Blue-Depth (RGB-D) cameras, specifically the Microsoft Kinect®, in Section II-B. We will also discuss work on mesh refinement in Section II-C.

A. Multi-View Reconstruction and Structure-from-Motion

The field of computer vision includes sub-fields such as object detection, tracking, and multi-view scene reconstruction. The goal of multi-view scene reconstruction is to extract a 3D point cloud representing a scene when given multiple views (such as photographs) of the scene. Seitz et al. provide a detailed analysis and comparison between various popular methods [7]. Most of these methods seek to create correspondences between views (images), usually by detecting features and tracking them from view to view. One of the main algorithms used to create the point correspondences is Scale-Invariant Feature Transform (SIFT) [8]. This collection of feature matches for a pair of images for a given feature across all images is known as a feature track. The point correspondences are used to compute estimations for the pose of the camera at the time it took each image. With this data, we can triangulate the locations of our tracked features to create a 3D point cloud. The density of this point cloud depends completely on the number of features tracked, which can be very few if the scene lacks interesting textures, varies is lighting condition, or is non-static. Finally, one typically meshes the resulting point cloud. For an excellent overview of many classical vision algorithms, the reader is referred to Hartley and Zisserman [9].

A complete multi-view reconstruction pipeline is available through a freeware program, VisualSfM [5], [6]. It utilizes bundle adjustment for error correction [10] and Patch-based Multi-view Stereo to densify spare point clouds [11].

B. RGB-D Cameras

There has been interest in utilizing depth sensing technologies for object reconstruction for a long time [12], but it is only recently that the technology has become very affordable and easy to use with the release of the Microsoft Kinect® in late 2010. With its release, came a plethora of reconstructions of people [13] and indoor environments [14]. The Kinect allows for the capture of 640 by 480 resolution depth images, and 1280 by 960 resolution color images. The depth data is derived from a structured light scheme using an infrared emitter and sensor.

One of the most important algorithms that has enabled reconstruction research is the KinectFusion [4], which fuses depth data and color data from a Kinect in real time to create a dense scene reconstruction as the user moves through the scene. In short, KinectFusion works by ray tracing (no bounces) through each depth pixel on the image plane by a distance equal to the depth, at each frame, and creating a vertex there.

C. Mesh Refinement

There are a plethora of mesh modification algorithms, but we focus on mesh generation and mesh decimation. The ball-pivot algorithm [15] is a mesh generation technique that fits our purposes well because we specify the radius of the ball that is effectively going around and connecting vertices. This allows our initial mesh to be as complex as we wish. While the KinectFusion data comes meshed, it is unnecessary complex and it should be remeshed using ball-pivoting after subsampling to create a point cloud. We recommend and use Poisson Disk Sampling [16]. Ball-pivoting can be used directly on VisualSfM point cloud data. While many mesh decimation techniques exist [17], we will utilize a generalized representative algorithm, Quadric Edge Collapse Decimation (QECD) [18]. QECD allows one to specify a target number of faces as an input parameter, which is, again, highly desirable for creating meshes of arbitrary complexity.

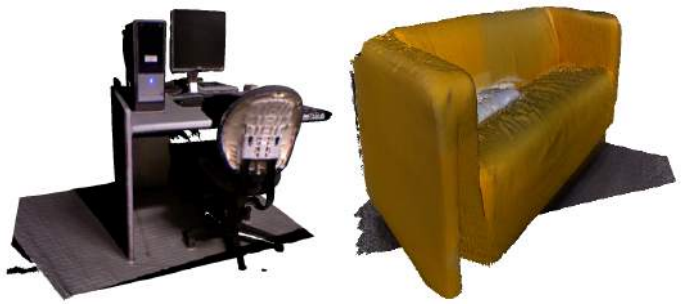
III. Methodology

KinectFusion capture was performed on a Dell Inspiron N7110 with an Intel Core i7-2630QM, 8GB of RAM, and GeForce GT 525M on Windows 7. Refinement was performed on a Alienware Aurora-R3 with an Intel Core i7-2600, 16GB of RAM, and a GeForce GTX 580 on Windows 7. We used Kinect for XBox with the official SDK, version 1.8. When performing KinectFusion reconstructions, no special considerations were made (lighting conditions, time of day, etc...).

A. Data Acquisition

The first data acquisition method we wanted to test was the Kinect and it's KinectFusion algorithm. We used the standard implementation of KinectFusion that comes with the Microsoft SDK. Some of our reconstructions are in Figure 2.

We found that the program was relatively easy to use and allowed us to produce compelling and complete reconstructions of our immediate surroundings in less than 20 minutes. Furthermore, the dense nature of the reconstructions reduces the need to fill holes in the mesh or add to the mesh in other ways.



(a) (b)

Fig. 2. KinectFusion reconstructions of a computer workstation with 1.7 million faces (a) and a couch with approximately 802,000 faces (b).

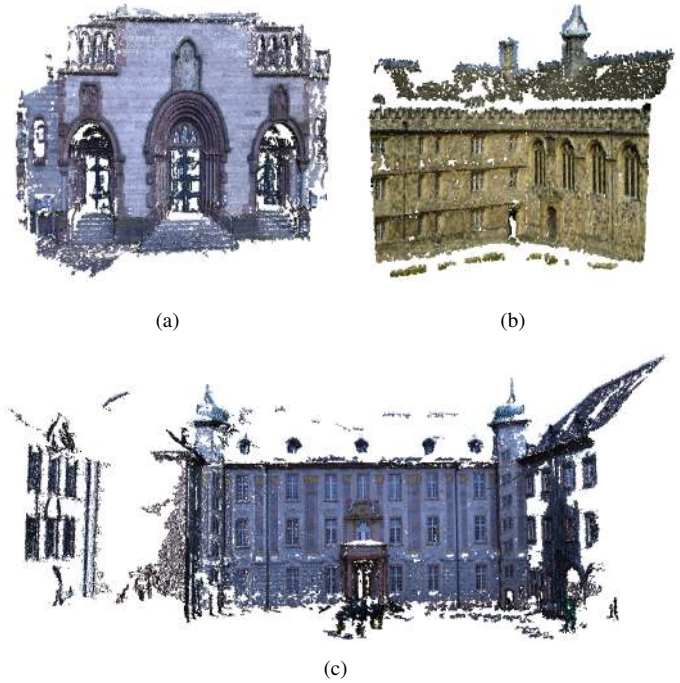
Although the result is generally appealing, there are a host of limitations to consider. First, the default KinectFusion algorithm is meant only for reconstructing the immediate area. In particular, the optimal distance for good depth data is 1.8

meters (although support for 0.8m to 4m exists) [19], and the way the algorithm works prevents you from moving into other areas. Although spatially extended upgrades exist, such as Kintinuous [20], we were unable to achieve satisfactory results with similar open-source implementations (Kinfu [21]). As an alternative, we stitched together the meshes by manually matching key points, like corners. For example, the cubicle in Figure 4 is the combination of seven meshes. MeshLab was used for the stitching. There are two additional large limitations to consider. First, the method is only viable in indoor environments due to infrared interference (although, it has been shown that data can be captured during sunset [22] . As such, we present an alternative with the computer vision program, VisualSfM. Second, the initial result is overly complex, with simple walls and floors consisting of hundreds of thousands of edges. Techniques to reduce complexity in Section III-B.

As an outdoor-capable alternative, we tried structure-from-motion using VisualSfM. We used various image datasets of interesting outdoor locations [23], but indoor environments are viable as well. Some results are in Figure 3.

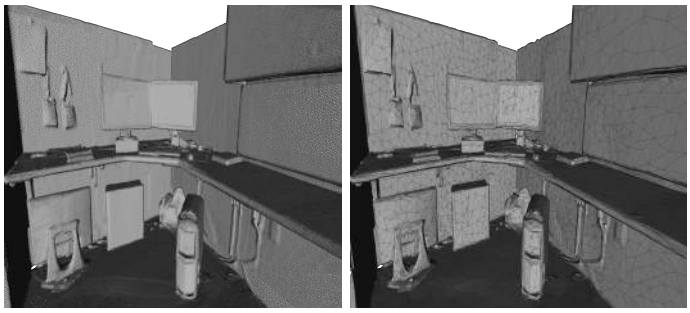
For these heavily textured scenes, the results are quite satisfactory, requiring stitching and minor clean-up of noise and reflective areas like windows. The picture taking process can take longer than Kinect if you're taking the photos yourself, but one big advantage is that you can utilize existing datasets from anywhere, including collections of tourist photos from Flickr (http://flickr.com). Furthermore, the results can pick up on small details with greater accuracy than the Kinect, which is limited by the resolution and noise of its depth sensor.

Although results are compelling for these datasets, many limitations exist. First of all, textureless places, like white walls, are not going to be picked up by this method because no SIFT features will be detected. This same limitation causes a greater number of holes and less density in the resulting mesh as opposed to KinectFusion. Second, the method can be sensitive to fluctuations in lighting conditions. In particular, it has been shown that while a small amount of data with lighting variation can be highly problematic [24], VisualSfM is capable of filtering large datasets with lighting variations (day and night) to produce good models [25].



(a) (b) (c)

Fig. 3. VisualSfM reconstructions of Herz-Jesu (a), Oxford's Wadham College (b) and a castle (c).



(a) (b)

Fig. 4. Mesh view of a cubicle scene with nearly 9.5 million faces (a), and a simplified version with only 50,000 faces (b).

B. Data Refinement

For data refinement, we used MeshLab. After using the ball-pivot [15] algorithm to create our initial meshes, we realized that the number of faces was simply too large for use in a game. Using Quadric Edge Collapse Decimation [18], we can simplify the scene to a desirable target number of faces. In Figure 4, we see that a cubicle scene with 9.5 million faces is unneccessarily complex (so much so that the individual triangles are too small to be seen), but can be simplified, with minimal quality loss, to 50,000 faces. Note that this algorithm is indiscriminate when it comes to simple areas, like walls which really only require one quad, and complex areas. Optimal decimation schemes should include additional algorithms, such as plane fitting [26].

It should be noted that due to how KinectFusion works, only areas in the immediate field of view at the start of the algorithm are scanned. To overcome this, we stitched several meshes together by using MeshLab's keypoint-matching method. No stitching is necessary for VisualSfM.

C. Data Usage

Finally, we prove that these meshes can be used to rapidly prototype a game using Unity. We placed several of our resulting meshes in a scene, had Unity create mesh-based colliders for them, and added a stock 3rd person player asset. In mere minutes, we can interactively explore the environment with proper collision detection.

Furthermore, Unity itself can be used for further mesh modification. For example, in Figure 5, we swapped out the material specification for our couch to make it green, elongated the couch along only one axis, and scaled it to realistically match the character's height.

IV. Conclusion and Future Work

In this paper we've shown that a low-cost workflow exists for digitizing real-world environments for use in real-time systems like games. In particular, we recommend using the KinectFusion for indoor environments, especially when just

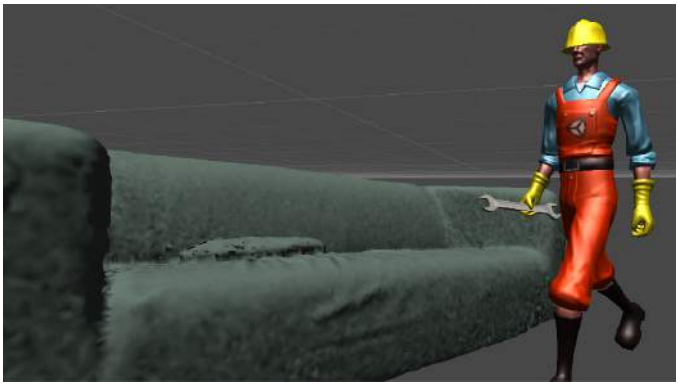


Fig. 5. A recolored couch being experienced inside Unity by a stock 3rd person player asset.

reconstructing objects or small environments. MeshLab can be used to stitch together meshes for large environments and also to simplify the mesh using Quadratic Edge Collapse Decimation so that the complexity is feasible for games. The resulting mesh lends itself nicely to a "subtraction" refinement, where the level or asset designer should remove noise and chisel in details from the overly smoothed mesh. For outdoor environments, we recommend photography and Visual SfM which can allow you utilize tourist photographs. The relatively sparse resulting mesh lends itself more to an "additive" refinement, where the designer should use the highly accurate point cloud to fit their own planes and use the colors for texture inspiration.

In the future, we would like to examine another outdoor alternative, LiDAR, which is higher cost but theoretically better. Also, we would like to directly compare the accuracy of Kinect, VisualSfM, and LiDAR, but it is an challenging task and out of the scope of this paper. Furthermore, we would like to explore specialized mesh decimation techniques that are specifically tailored to ease the workload of level and asset designers.

REFERENCES

[1] "Gran Turismo 5," Sony Computer Entertainment, 2010.

[2] J. Greer and Abraxas. Gran Turismo 5 vs. Reality: Can You Tell the Difference? Accessed: June 2014. [Online]. Available: http://www.gtplanet.net/gran-turismo-5-vs-reality-can-you-tell-the-difference/

[3] B. Preston. GT6: How Video Games Can Influence Sports Car Design. Accessed: June 2014. [Online]. Available: http://wheels.blogs.nytimes.com/2013/09/25/gt6-how-video-games-can-influence-sports-car-design/

[4] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon, "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*, ser. UIST '11. New York, NY, USA: ACM, 2011, pp. 559–568. [Online]. Available: http://doi.acm.org/10.1145/2047196.2047270

[5] C. Wu, S. Agarwal, B. Curless, and S. Seitz, "Multicore bundle adjustment," in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, 2011, pp. 3057–3064.

[6] C. Wu, "Towards linear-time incremental structure from motion," in *3DV-Conference, 2013 International Conference on*, 2013, pp. 127–134.

[7] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski, "A Comparison and Evaluation of Multi-View Stereo Reconstruction Algorithms," in *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 519–528.

[8] D. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal On Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[9] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, 2004.

[10] M. Lourakis and A. Argyros, "The design and implementation of a generic sparse bundle adjustment software package based on the Levenberg-Marquardt algorithm," Institute of Computer Science - FORTH, Heraklion, Crete, Greece, Tech. Rep. 340, August 2000.

[11] Y. Furukawa and J. Ponce, "Accurate, dense, and robust multi-view stereopsis," in *IEEE Conference on Computer Vision and Pattern Recognition*, June 2007, pp. 1–8.

[12] Y. Chen and G. Medioni, "Object modelling by registration of multiple range images," *Image Vision Comput.*, vol. 10, no. 3, pp. 145–155, Apr. 1992. [Online]. Available: http://dx.doi.org/10.1016/0262-8856(92)90066-C

[13] J. Tong, J. Zhou, L. Liu, Z. Pan, and H. Yan, "Scanning 3d full human bodies using kinects," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 18, no. 4, pp. 643–650, 2012.

[14] A. Majdi, M. C. Bakkay, and E. Zagrouba, "3d modeling of indoor environments using kinect sensor," in *Image Information Processing (ICIIP), 2013 IEEE Second International Conference on*, 2013, pp. 67–72.

[15] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin, "The Ball-pivoting Algorithm for Surface Reconstruction," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 5, no. 4, pp. 349–359, Oct 1999.

[16] M. McCool and E. Fiume, "Hierarchical poisson disk sampling distributions," in *Proceedings of the Conference on Graphics Interface '92*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1992, pp. 94–105. [Online]. Available: http://dl.acm.org/citation.cfm?id=155294.155306

[17] M. Garland, "Multiresolution modeling: Survey & future opportunities," in *Eurographics '99 – State of the Art Reports*, 1999, pp. 111–131. [Online]. Available: http://graphics.cs.uiuc.edu/\~{}garland/papers/STAR99.pdf

[18] M. Garland and P. S. Heckbert, "Surface simplification using quadric error metrics," in *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '97. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997, pp. 209–216. [Online]. Available: http://dx.doi.org/10.1145/258734.258849

[19] Kinect Sensor Troubleshooting. Accessed: June 2014. [Online]. Available: http://support.xbox.com/en-US/xbox-360/kinect/sensor-placement

[20] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald, "Kintinuous: Spatially extended KinectFusion," in *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, Sydney, Australia, Jul 2012.

[21] KinectFusion Extensions to Large Scale Environments. Accessed: June 2014. [Online]. Available: http://www.pointclouds.org/blog/srcs/fheredia/index.php

[22] R. El-laithy, J. Huang, and M. Yeh, "Study on the use of microsoft kinect for robotics applications," in *Position Location and Navigation Symposium (PLANS), 2012 IEEE/ION*, April 2012, pp. 1280–1288.

[23] C. Strecha, W. von Hansen, L. V. Gool, P. Fua, and U. Thoennessen, "On Benchmarking Camera Calibration and Multi-View Stereo for High Resolution Imagery," in *CVPR'08*, 2008.

[24] S. Recker, M. M. Shashkov, M. Hess-Flores, C. Gribble, R. Baltrusch, M. A. Butkiewicz, and K. I. Joy, "Hybrid photogrammetry structure-from-motion systems for scene analysis and measurement," in *Coordinate Metrology Systems Conference*, 2014.

[25] N. Snavely, S. M. Seitz, and R. Szeliski, "Modeling the world from internet photo collections," *Int. J. Comput. Vision*, vol. 80, no. 2, pp. 189–210, Nov. 2008. [Online]. Available: http://dx.doi.org/10.1007/s11263-007-0107-3

[26] M. Y. Yang and W. Forstner, "Plane detection in point cloud data," in *Proceedings of the 2nd int conf on machine control guidance, Bonn*, vol. 1, 2010, pp. 95–104.