Novel Techniques for Processing Building Exteriors Captured from Photogrammetry

Scott Johnson, Scot Shiflett
Leidos, MS&T Division
Orlando, FL
Scott.johnson-4@leidos.com
scot.b.shiflett@leidos.com

Clayton Burford
US Army STTC
Orlando, FL
clayton.w.burford.civ@army.mil

ABSTRACT

The One World Terrain (OWT) project seeks to create a digital model of the entire world for use in US Army simulators. As part of an ongoing research effort within DEVCOM Soldier Center Simulation & Training Technology Center, High-Resolution Insets are being added to the overall terrain inventory using photogrammetry-based methods with drone-captured photos and LiDAR data. A pipeline of processing stages operates on the possibly thousands of drone-captured photos of an area to create the terrain content. A crucial stage of the pipeline involves identification, extraction, and optimization of building exteriors for subsequent use. Currently, the building-processing stage produces reference material for artists to manually create accurate 3D models of the buildings. Simultaneously, efforts are underway to fully automate the building-processing stage. This paper will acquaint the reader with the unique output of the photogrammetry process, highlighting the large gap between the raw output and what is needed for today's simulators. The paper then focuses on several novel techniques within the building-processing stage that eliminate noise and reduce complexity. Finally, the paper will demonstrate current results of the entire automated building pipeline.

ABOUT THE AUTHORS

Scott Johnson is a contractor for Leidos. Previously he was a Principal Engineer at Magic Leap and the Chief Technology Officer at Chosen Realities doing Augmented Reality. Scott has 19 years of experience in defense and 10 years of experience in the video game industry. He earned his Bachelor of Science in Electrical Engineering from Purdue University and a Masters in Computer Science and Engineering from the University of Michigan.

Scot Shiflett is a Project Manager at Leidos for the Leidos Orlando Geospatial Innovation Center (LOGIC) lab. Mr. Shiflett manages the resources associated with program activities for database requirements development, source data collection and standardization. Mr. Shiflett holds a Bachelor of Science in Industrial Design from Auburn University.

Clayton Burford is the Science & Technology Manager for digital terrain research and development at the U.S. Army's DEVCOM Soldier Center. His professional experience includes over 15 years of Department of Defense acquisition. His focus is on digital terrain content applicable to Army Warfighting Functions with an emphasis on its utility within Training Aids, Devices, Simulators, and Simulations (TADSS).

Novel Techniques for Processing Building Exteriors Captured from Photogrammetry

Scott Johnson, Scot Shiflett
Leidos, MS&T Division
Orlando, FL
Scott.johnson-4@leidos.com
scot.b.shiflett@leidos.com

Clayton Burford
US Army STTC
Orlando, FL
clayton.w.burford.civ@army.mil

PROBLEM

Techniques to create terrain for simulators have been used for decades. The traditional way is to generate the terrain polygons from the abstract Geographic Information System (GIS) data such as linears for roads, areals for lakes, and Digital Elevation Model (DEM) data for the height of the terrain. The more modern way is to capture photographs of the intended area and then feed them into photogrammetry software to create the terrain polygons. Then additional software needs to find the abstractions back from the generated polygons. So, the modern way is the opposite of the traditional way. This creates an opportunity for unique and novel stages in the processing pipeline.

In the big picture, our overall project creates high resolution terrain from photogrammetry. The process begins when a commercial drone is flown around the US Army training areas and photographs are taken as a part of a precise capture process (Lam et al., 2023). A previous analysis of available photogrammetry software concluded that Bentley's iTwin Capture (formerly Context Capture) best meets the project's needs. All site-captured photographs are fed into iTwin Capture, which generates a set of textured triangle meshes. This paper focuses on the automated processing of this output to make building exteriors suitable for use in various simulators. In addition, we currently have a team of artists that manually model the building's exteriors and interiors. The project's goal is to assist the artists and ultimately make an automated pipeline for the entire process. Part of the assistance includes providing the artists with simplified geometry and extracted textures from the raw captured buildings. The pipeline process can be rather elaborate because of the unique challenges of ingesting iTwin Capture buildings, as they are full of mesh anomalies. The other challenge is to process and understand the characteristics of the buildings such that their shapes can be represented in a minimal way. This will create low detail representations of the meshes that can be used by themselves or as an initial aid for the artists to start modeling. This paper describes some of the key stages of the automated modeling process and shows intermediate and final results.

Mesh Anomalies in iTwin Capture Buildings

The first challenge in creating a model pipeline is to fix the tile boundaries that are present in the iTwin Capture terrain output. iTwin Capture generates the entire terrain as separate tiles whose boundaries overlap. Figure 1 shows a patch of terrain with a building from Fort Polk. This building has multiple tile boundaries (shown in red) that fall across its surface. The intersecting faces in the mesh are displayed as red bands. In this example, there were 162,000 pairs of intersecting faces.

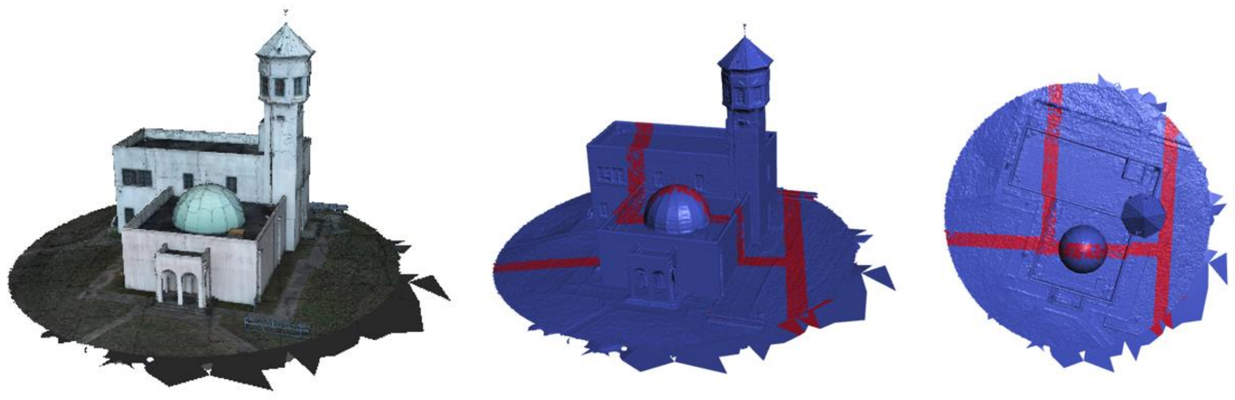

Figure 1. A section of terrain from Fort Polk that has a building. The red stripes are iTwin Capture's bands of intersecting triangles that must be fixed.

Correcting the tile boundaries is just the beginning. The iTwin Capture photogrammetry process results in a large amount of erroneous geometry. Figure 2 shows the St. Francis Hospital from the Muscatatuck Training Center and its surrounding terrain. The erroneous geometry below the terrain surface is referred to as a "sinker", while the erroneous geometry above the terrain surface is a "floater". The example of Figure 2 has a single floater in the upper left of the figure and a large sinker below the building. Terrain viewers hide the geometry beneath the terrain skin so usually no one knows it is there. But any processing of these buildings must be able to remove the triangles of the sinkers and floaters.

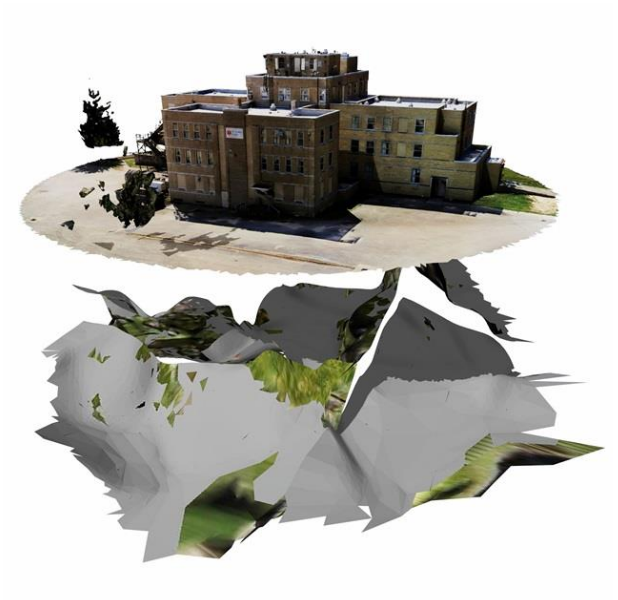

Figure 2. Extracted iTwin Capture mesh of the St Francis Hospital at the Muscatatuck Training Center. There is a large amount of erroneous geometry beneath the terrain skin.

Towards that goal, we would like to take advantage of the many powerful libraries that exist for processing meshes. The problem is that the iTwin Capture meshes have so many anomalies that they will not load into libraries that require a clean connected surface. In fact, the meshes demonstrate most of the anomalies in what Botsch (Botsch, et al., 2010) calls "The Freak Show". The meshes have holes, singular vertices, intersecting faces, handles, and faces with inconsistent orientation. At a minimum, the meshes need to be cleaned up enough so that they will load into the Polygon Mesh Library and be treated as a manifold.

This paper focuses on three of the main stages of the solution: Remeshing, Face Segmentation, and Retexturing. The final stage of stitching together the faces of the output buildings will only be covered briefly.

REMESHING

The purpose of this stage of the pipeline is to clean up the iTwin Capture meshes and get them as close as possible to manifold. This will reduce the burden of the anomalies on all the later stages of the processing pipeline. We start by considering existing work in Remeshing.

Previous Work in Remeshing

Poisson Surface Reconstruction (Kazhdan et al., 2006) takes a point cloud and fits a surface to it. An interactive application called Mapple (Nan, 2017) allows users to load meshes, resample them to point clouds, and run Poisson Surface Reconstruction. We quickly learned that the tile boundaries and sinker geometry often overwhelmed this algorithm. It was also sensitive to a set of input parameters that would have been challenging to fix in a noninteractive environment.

Easy3D (Nan, 2017) is a set of mesh processing tools that include cleaning polygon meshes. It has operations to detect and remesh intersecting faces. It was best at fixing nonmanifold vertices and reporting the number of vertices it was able to fix. Easy3D's operations were able to remove many of the standard mesh anomalies, but none could handle the tile boundaries.

Blender (Blender Foundation, 2024) is a free modelling tool used by 3 million users worldwide. Just one of its many uses is to model and edit meshes. Blender includes a mesh modifier called "remesh" that voxelizes the mesh and creates a new set of faces from the voxels. Its implementation is a straightforward use of the open-source library for voxels called OpenVDB (Museth et al., 2013). Blender converts its mesh to an OpenVDB mesh. Then OpenVDB is used to render the mesh into voxels, where each voxel records a distance from its center to the surface created by the triangles of the mesh. Lastly, new triangles are extracted from the voxels by finding where the distance to the mesh surface crosses zero.

The catch is that in many cases Blender remeshing does not work. Users all over the world have loaded their model into Blender, hit "remesh" and their mesh turns into a few blobs. Blender's secret is that its remeshing only works on closed meshes. Unfortunately, the terrain patches that contain our buildings are not closed meshes. (They are homeomorphic to a disk). But this does lead in the direction of a solution.

Custom Remeshing

Why does Blender's remeshing have this limitation? OpenVDB is the standard voxel library in visual special effects and won an Academy Award for Technical Achievement in 2014. OpenVDB should prominently state that it is intended for closed meshes, but it does not advertise this limitation. It will render non-closed meshes to voxels, but it will not create a signed distance from the voxels to the mesh surface, only an unsigned distance. The signs need to be corrected for all the voxels when using a non-closed mesh. The triangle normals can be used to determine the sign of the distances. The problem is that we already know that there are flipped triangles in the mesh. We must defensively limit damage to our remeshed triangles from the bad normals.

The functionality of OpenVDB we needed to modify was embedded too deeply in their code for successful reuse. A custom K-D Tree was made of the mesh triangles. Using OpenVDB, we created a band of voxels around the mesh surface with enough voxels away from the surface such that we could iterate across the voxels of the surfaces but stay away from overall background value. OpenVDB only stores the voxels in a band near the mesh surface. Anything outside that band gets the signed distance of the background value. Therefore, our algorithm carefully iterated in the narrow band of voxels to avoid querying the background area. For each voxel center, the eight nearest neighboring faces are found in the K-D Tree. Then we process the normal of each neighboring face and use a voting scheme to determine the most likely normal direction. Outliers are removed and a plane is fit to the remaining face centroids. The plane normal is the normal to the surface at the query point. Thus, we have defensively limited the problem of a few flipped triangles. The correct normal allows us to fix the sign of the distance already created by OpenVDB. We now have a set of voxels with correct signed distances to the mesh surface.

Next, the new faces from the corrected voxels must be extracted. OpenVDB had an implementation of Marching Cubes (Lorensen, 1987) for this purpose. But it was customized to only work with closed meshes. Even with our sign corrected voxel set, OpenVDB double sided the level set. It was determined to output a closed mesh. Though it seemed that it would be trivial to change, there was no good way to alter this behavior in the existing source code. We were already extensively using the Geometric Tools Engine (Eberly, 2000) in our model pipeline. We adapted its solution for Marching Cubes to use OpenVBD voxels instead of fully populated 3D array.

At this point we have met our goal. We input the iTwin Capture meshes to our Remeshing algorithm and it results in a surface mesh from the Polygon Mesh Processing Library. We then traverse the connected triangles and throw out all but the largest connected set. The process removes all but the toughest floaters and sinkers. It also opens the doors to all the algorithms available in the Polygon Mesh Library such as decimation, triangulation, and parameterization.

Remeshing Results

We remesh the buildings and their surrounding terrain patches to a voxel size of 5cm. The tile boundaries are cleanly removed. It eliminates all the intersecting triangles and all the nonmanifold vertices. Marching Cubes is known to add some of its own artifacts. Some surface anomalies remain after this step which are frequently due to an overwhelming number of bad normals in the input mesh. Many manifest as discontinuous geometry that are removed in the next step in the pipeline. The iTwin Capture mesh from Figure 1 started with 162,000 pairs of intersecting faces. The remeshing removes all of them at the cost of rounding the edges in the building. An additional cost is that since we voxelized to such a small size (5cm), the face count in that example nearly doubled to over a million faces. The latter stages of the pipeline must handle face counts in the range of one million faces.

One reason we consider this stage to be novel is not only that it eliminates the tile boundaries and so many of the iTwin Capture mesh anomalies, but it does it in a way that is sharable to millions of Blender users. We could submit our work to Blender so that this work can be used outside of our project. In short, we have the solution to a problem that Blender users frequently encounter.

FACE SEGMENTATION

The next novel stage is the process of segmenting the faces into groups that have some meaning. One such segmentation is to separate the faces of the building from the faces of the ground terrain. Another possible segmentation is to break the mesh into walls so that the entire wall is recognized rather than its thousands of triangles. Note that the input to this stage is the output mesh of the Remesh stage.

Again, we start from previous work. IsoChart (Zhou, et al., 2004) is published by Microsoft with code for the purpose of segmenting models. Its goal is to segment the faces into “charts” that are contiguous patches of triangles in UV space. When running it on our remeshed buildings it created clean charts, but they semantically had nothing to do with the building. We want a partitioning that breaks the faces at edges of high curvature. While it did segment our buildings, the segments were not useful for understanding the building shape.

Liu, et al. (Liu et al., 2004) uses spectral clustering to break the faces into segments. A key component of the method is to compute an affinity matrix W_{ij}, that describes the likelihood that faces ‘i’ and ‘j’ are in the same segment. Our input mesh (the result of remeshing) can contain 1 million faces. That would make a matrix that is 1 million rows by 1 million columns. Even with sparse matrix techniques, this is too much to ask of the solution.

Katz et al. (Katz et al., 2003) is frequently cited, yet it has a similar limitation. The first step in its algorithm is to compute the distance from every face to every other face. Its test models are very simple models that avoid this obvious $O(n^2)$ problem. But inside that paper is an overall approach that is very useful to our problem. They create a dual graph of the mesh faces such that each face in the mesh becomes a node in the graph and each edge in the graph represents an edge between two neighboring triangles. Then they weight the edges in the graph with a capacity that is based on the dihedral angle between the faces. Figure 3 illustrates a dual graph of a mesh of four connected triangles. Each triangle becomes a node in the graph. In this example, the source node is connected to two of the faces and the sink node is connected to three faces. Katz et al uses the maximum flow minimum cut to determine a partitioning of each triangle to either source or sink based on an imagined flow from source to sink. See (Kleinberg et al., 2005) for more details.

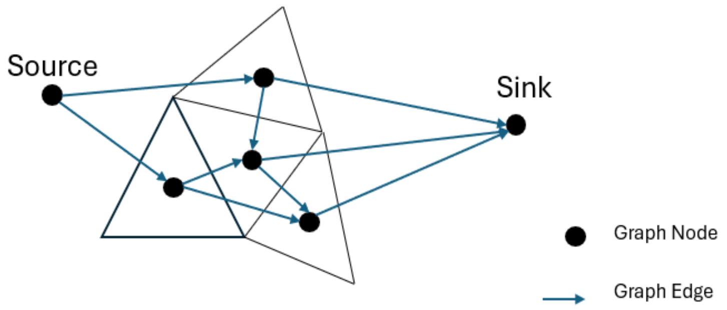

Figure 3. A Triangle Mesh and its Dual Flow Graph.

The overall approach of using an initial segmentation then refining it using maximum flow minimum cut is directly applicable to buildings. We use RANSAC plane fitting (Schnabel et al., 2007) as the initial partitioning, then we refine it using maximum flow minimum cut. The input to the plane fitting is a point cloud. Our highly dense resampled mesh is sampled into a point cloud trivially by using the triangle centroids and triangle normals as the points in the point cloud. The algorithm requires the choice of five parameters, including the minimum number of points on the plane to make a new segment. Parameters add risk to a model pipeline because there is no guarantee that one will ever find a set of parameters that works across all the inputs. Tests have shown that this step works well and is not very sensitive to its parameters.

After fitting the mesh to planes, we need to use maximum flow minimum cut to refine the results. Katz (Katz et al, 2003) used the Ford Fulkerson algorithm for maximum flow minimum cut. The underlying algorithm does a Dijsktra's Shortest Path from the source node to the sink node pushing the maximum capacity of the found path through the graph edges at every stage. It terminates when it cannot push any more capacity though the edges. At each iteration, the whole path from source to sink is thrown out and another one is started. This is a big hint that it will not scale to several million nodes. Not surprisingly, our implementation ran correctly on small buildings, but was miserably slow on the larger buildings. We had to move to the more powerful (Boykov & Kolmogorov, 2004) implementation.

For ground segmentation, we made use of a mesh that was made from a previous step in the overall terrain pipeline that creates a bare earth mesh for the terrain. This is a terrain mesh without trees and buildings or any other culture. We decided that each terrain face that had an upright normal and was close to the bare earth would be connected to the source in the flow graph. Then we fit the rest of the mesh to planes and decided which planes were likely to be walls. Walls are upright and are generally at least 10 feet tall. The faces that fit to wall planes were connected to the sink in the flow graph. The rest of the flow graph was created with flow according to Katz et al (Katz et al., 2003). We ran the maximum flow minimum cut on hundreds of thousands of faces. In less than a second, we get an assignment of each face to either the source or sink groups. We can then throw out the source group because those are the ground faces.

Face Segmentation Results

Results are shown in Figure 4. The blue faces are the ground faces. The gold faces are the nonground faces. The seam between the two groups is where the maximum flow between the source and sink nodes occurred. This seam can be turned into a 3D footprint for the building that allows it to be stitched into the terrain.

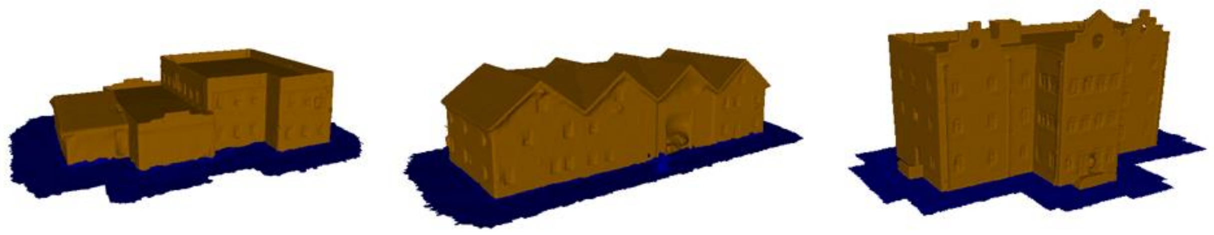

Figure 4. Ground Segmentation results. Blue faces are ground faces. Gold is not ground.

After removing the ground faces, we are ready for the next pass of segmentation. We refit the remaining faces to planes. This creates a set of partitions in the mesh. Since the max flow algorithm can only differentiate between two groups: source or sink, we must apply it to each segment in turn. The source is connected to all the faces in the current plane. The sink is connected to all the other nonpartitioned faces so far. The process iterates over all the fit planes, one at a time. Results are shown in Figure 5 for several buildings. The colors are per face in the model. Faces that have the same color are in the same segment. This partitioning is close to following the walls and roof surfaces of the building. This is a key step that leads to understanding of the overall shape of the building.

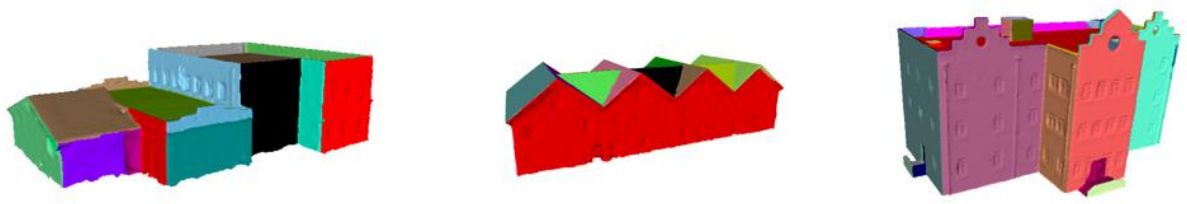

Figure 5. Face Segmentation results. The triangles are in meaningful partitions.

Many readers may be wondering why we are not mentioning Artificial Intelligence (AI) or Machine Learning (ML) for face segmentation. Our team is currently working towards that goal. This current solution uses the bare earth mesh as an initial hint of what faces are ground. Our overall strategy of using an initial source for segmentation and then refining using maximum flow minimum cut is applicable for when AI-based segmentation is working in the pipeline. We just apply the AI segmentation as the initial segmentation and then refine it using maximum flow minimum cut to be per face. Also, the results of this algorithmic solution could provide excellent training data for the ML solutions.

RETEXTURING

As soon as the iTwin Capture meshes were remeshed, we took on the problem of putting the textures back because remeshing creates a new set of triangles without a UV map. To address this, we created code to Retexture the mesh from the original iTwin Capture textures. We had to create a new parameterization of the processed model to create a UV Map from the 3D vertices to 2D UV coordinates. We start by grouping the faces by their normals. Then we clean up the boundaries of the groups, again using maximum flow minimum cut. Then we run Least Squares Conformal Mapping (Lévy et al., 2002) on the groups to flatten them into UV space. Once we have a target UV space to copy to, we raycast into the source textured model with the original textures from iTwin Capture. For each pixel in the output texture, we use the triangle gradient equations to determine which 3D point is equivalent to its 2D UV coordinate in the destination mesh. Then we cast a ray from the destination triangle to the source mesh. Wherever it hits, we determine the pixel in UV space for that 3D hit point and bilinear filter the resulting pixel from the source texture.

Speed is important because the ray casting will be called millions of times. We used ray triangle intersection (Moeller, 1997) but wrote it using AVX2 SIMD vector processing. Measurements show the SIMD speeds up the process by 3 times and multithreading with 12 cores speeds it up 8 times. We atlas all the single textures together so that the artists do not have to open 50-100 textures one at a time in Photoshop if they choose to hand edit the results. This stage is called Texture Baking in various modelling tools. It is an important step in any model processing pipeline, yet we do not know of any open solutions for it. Plus, our artists prefer the results of our automatic texture baking over the one in Maya.

EXTRACTING TEXTURES

The stages of Remeshing, Face Segmentation, and Retexturing combine to greatly reduce the artist team's manual labor. Since the artists spend the largest amount of time making textures for the buildings, they asked for an application that could extract the major planes from the buildings and create a set of atlased textures. Using the stages already mentioned, we were able to extract the textures from the input iTwin Capture meshes. We had to remove the sinkers and floaters and then the ground faces so that none of those faces appear in the atlas. Figure 6 (left) shows four of the eight iTwin Capture textures that are used for the Muscatatuck Hospital building and its surrounding terrain. The artists used to search the input textures for the building surfaces. Now they can use the one texture on the right that has all the major surfaces of the building and no faces from the surrounding terrain. This has greatly sped up their workflow on large complex buildings.

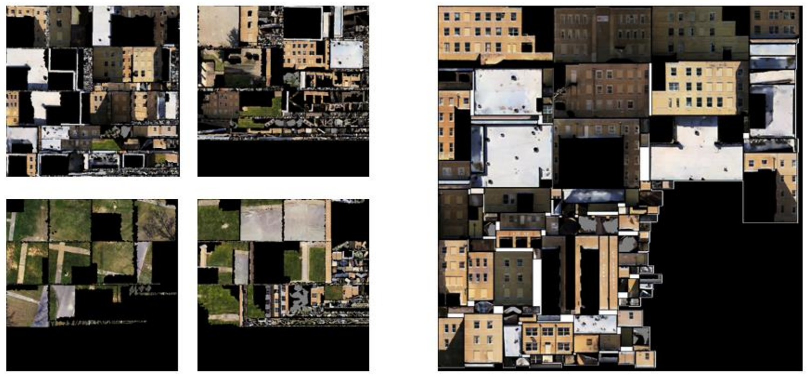
Figure 6. Compare the iTwin Capture Textures (left) that have terrain in them with the one automated texture made for the artists (right) that is just for one building.

BUILDING PIPELINE OUTPUT

The highlighted pipeline stages covered previously are key components of the pipeline but there is one stage left. The segmented faces are processed by finding the triangle edges that border different partitions. These edges are stitched together into cycles that will form the faces in the low detail mesh. Then the high detail model is searched for vertices which border three or more different partitions. All the vertices are removed from the stitched cycles except these special vertices. Then these cycles are inserted into code that makes a final model.

The results are good for certain types of buildings. For a building to process well, it must be mostly planar, it must be isolated from other buildings and nearby vegetation, and it must have been captured up close by the drone passes. Figures 7 and 8 demonstrate sample buildings which meet these criteria and the results of the full pipeline on these buildings. Future work intends to reduce the caveats in the pipeline for producing good buildings.

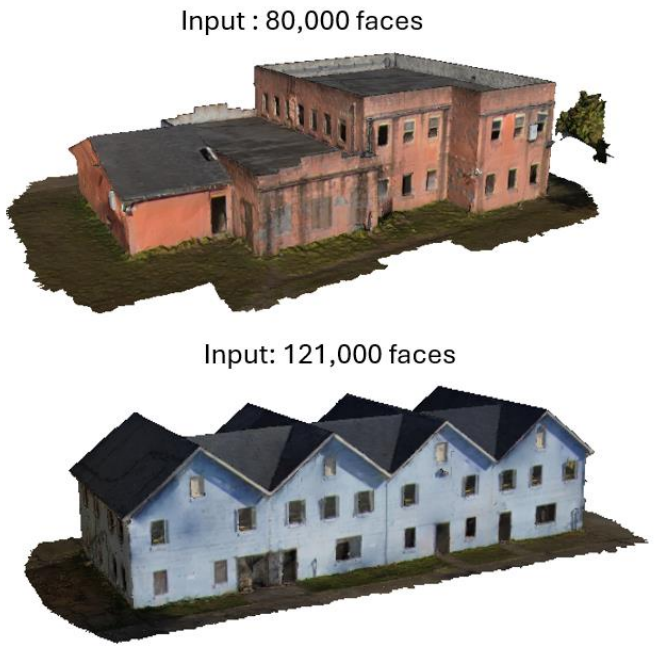

Figure 7. Two Building Results from the Fort Benning McKenna MOUT Site

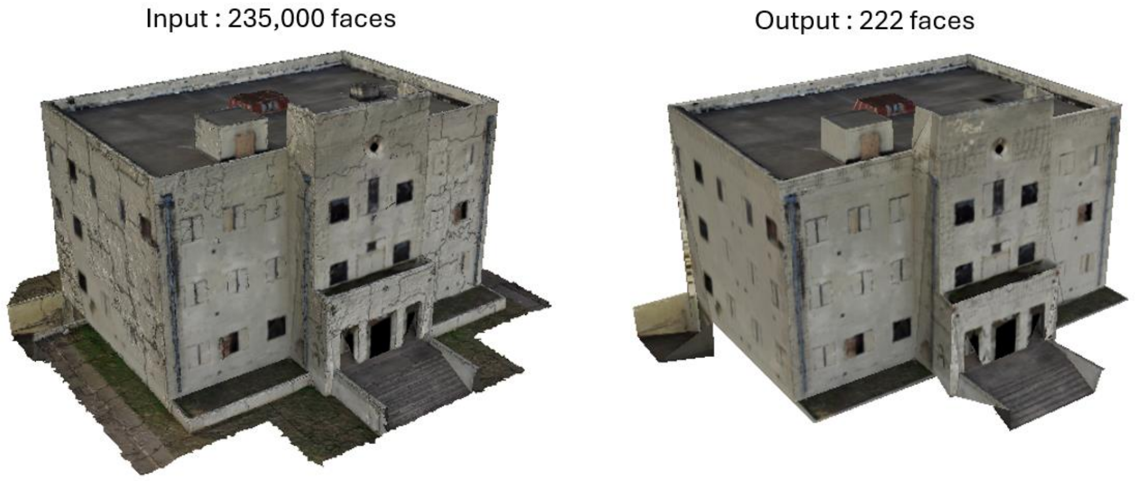

Figure 8. Building Results from Fort Campbell's Range 101

These buildings can be used as low detail models in various simulators as well as the initial reference material for our artists to make the highest detailed models by hand. They have a low face count as well as their own textures separate from the rest of the terrain. Without texturing, the models are low detail enough to be used for Live Training because they are a close approximation to the overall shape of the building.

CONCLUSION

We have introduced the difficult problem of processing terrain from photogrammetry and how it is opposite of the traditional process of generating terrain polygons. Then we showed how the combination of Remeshing, Face Segmentation, and Retexturing were used successfully in many ways. Firstly, we want to emphasize that our Remeshing solution can be directly integrated with Blender and millions of users can benefit from it.

The second success was to generate reference material for the art team to create precisely made buildings. The extracted textures and the cleaned geometry greatly sped up their work. The final success was to create an automated pipeline that creates low detail building meshes that can be directly used in some simulations. To our knowledge, these stages and how they are applied to building exteriors are unique.

REFERENCES

Blender Foundation. (n.d.). *Blender*. Retrieved February 14, 2024, from https://www.blender.org

Botsch, M., Kobbelt, L., Pauly, M., Alliez, P., & Lévy, B. (2010). *Polygon Mesh Processing*. AK Peters/CRC Press.

Boykov, Y., & Kolmogorov, V. (2004). An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 26*(9), 1124-1137. https://doi.org/10.1109/TPAMI.2004.60

Eberly, D. H. (2000-2024). *Geometric Tools Engine*. Geometric Tools, LLC. https://www.geometrictools.com/

Katz, S., & Tal, A. (2003). Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Transactions on Graphics, 22*(3), 954-961. https://doi.org/10.1145/882262.882370

Kazhdan, M., Bolitho, M., & Hoppe, H. (2006). Poisson surface reconstruction. *Proceedings of the fourth Eurographics symposium on Geometry processing*, 61-70.

Kleinberg, J., & Tardos, É. (2006). *Algorithm Design*. Pearson.

Lam, T., Reilly, M., Ramos, P. York, H. Larrieu, & M. Shiflett, S. (2023). Analyzing, Preparing, and Processing Input Geospatial Data for High-Resolution Terrain Generation. *Proceedings of the Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC), 2023*(1), 1-13.

Lévy, B., Petitjean, S., Ray, N., & Maillot, J. (2002). Least squares conformal maps for automatic texture atlas generation. *ACM Transactions on Graphics*, 21(3), 362-371. https://doi.org/10.1145/566654.566590

Liu, R. (2007). Segmentation of 3D meshes through spectral clustering. *Eurographics Symposium on Geometry Processing*, 115-123.

Lorensen, W. E., & Cline, H. E. (1987). Marching cubes: A high resolution 3D surface construction algorithm. *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '87),* 21(4), 163-169. https://doi.org/10.1145/37401.37422

Möller, T., & Trumbore, B. (1997). Fast, minimum storage ray-triangle intersection. *Journal of Graphics Tools, 2*(1), 21-28. https://doi.org/10.1080/10867651.1997.10487468

Museth, K. (2013). OpenVDB. *ACM Transactions on Graphics, 32*(3), 27:1-27:22. https://doi.org/10.1145/2487228.2487235

Nan, L. (2017) Easy3D. *Easy3D: A 3D graphics library*. Retrieved February 22, 2024, from https://3d.bk.tudelft.nl/liangliang/software.html

Schnabel, R., Wahl, R., & Klein, R. (2007). Efficient RANSAC for point-cloud shape detection. *Computer Graphics Forum, 26*(2), 214-226. https://doi.org/10.1111/j.1467-8659.2007.01016.x

Zhou, Q. (2004). Iso-charts: Stretch-driven mesh parameterization using spectral analysis. *Proceedings of the ACM SIGGRAPH Symposium on Geometry Processing*, Page numbers.