3D Modelling in Blender Based on Polygonal Data

James Ribe
MSCS Department
St. Olaf College
1500 St. Olaf Ave
Northfield, MN 55438
ribe@stolaf.edu

Alora Killian
MSCS Department
St. Olaf College
1500 St. Olaf Ave
Northfield, MN 55438
killian@stolaf.edu

Dan Anderson
MSCS Department
St. Olaf College
1500 St. Olaf Ave
Northfield, MN 55438
adersede@stolaf.edu

Abstract

We present a method of modelling a realistic interior of a building based on polygonal data from stereoscopic image pairs. By receiving the data from a previous stage in a pipeline, we use the 3D modelling program Blender to render and export a model into the game engine Irrlicht. We created importer and vertex merging tools in order to transport the data from raw information into .obj files. We used the .obj files to display a mesh in Blender, then exported it into COLLADA format for rendering in Irrlicht. We also suggest improvements and possible additions to better the process.

1 Introduction

Realistically modelling the interior of a building is a challenge in the field of computer vision and graphics. We were presented with the opportunity to participate in a college course attempting to fully render a model of St. Olaf College's Regents building. The project was a month-long endeavor involving a seven stage pipeline-- each one passing on relevant data to the next. Our team, Team LightsCameraRender, was the sixth stage in the pipeline. Our team's goal was to receive polygonal data and coordinates, import it into the program Blender, apply strategic lighting and reflectance values, then export the model into a format suitable for the game engine Irrlicht. The team following ours then uses Irrlicht to display our model in a 3D format.

1.1 Background

The success of the project, and our eventual 3D structure, depends on the existence of good, stereoscopic pairs of images. Other research on successfully modelling large objects does not use stereoscopic imaging to create their final structure, but instead uses overlapping images. Researchers from Hong Kong, for instance, used cameras attached to the tops of cars to capture Hennepin Avenue in Minneapolis, Baity Hill Drive in Chapel Hill, and Dishifu Road in Canton [9]. Their method decomposed photos of the streets into simple geometric forms and texture maps, and produced a model of the exteriors of buildings with a minimal amount of polygons and flat textures. Our method, on the other hand, hoped to create a model of the interior of our science building, which contains more polygons resulting in an extremely complex model. Another team of researchers created a 3D model of famous architecture, pulling thousands of images off of Flickr [1]. In contrast to this, our project had a specific team of photographers, who calculated the coordinates of the camera and also captured exact stereoscopic pairs of images.

In order to model the data we receive, we use the open source 3D model creator Blender. The program is generally used by artists sculpting a mesh for use with animation, character creation, or personal use [4]. Hand modelling is used extensively, and the technique can also be used to sculpt highly realistic objects on the caliber we aim to achieve with our model [6]. However, we do not use this method because it would take an inconvenient amount of time to realistically hand-model an exact duplicate of the Regents building. Our methods require only a minimal amount of hand-modelling --in order to delete minor artifacts-- due to the time limit constraints on our project. We use Blender to model Regents because it is a convenient program with a large repertoire of useful tools. There is little-to-no current research in realistically rendering a 3D model from images using Blender. We hope that our paper will be useful for further research on creating meshes in Blender based on stereo imaging and polygonal data.

1.2 Overview

In this paper we present our methods on successfully interpreting data received from other teams, and rendering it in an accurate manner. We discuss the three different stages in order to effectively create a large mesh. In section one, we present what is required to import 3D data in the form of an .obj file. In section two, we discuss what necessary processes we must go through in Blender, as well as creating texture and specularity maps. In section three, Irrlicht is analyzed and the details of correctly exporting the mesh are explained. Finally, we conclude with what research we hope will continue with Blender in the future.

2 Importing Files

After receiving data from earlier stages in the pipeline, our task was to turn raw information into polygons that could be imported into Blender. Because Blender has a plethora of various import formats, we chose to translate the initial data into the wavefront .obj format, due to its easy syntax and simple conversion properties.

The data we received contained source images, polygon vertex lists in the image coordinate space, and polygon vertex lists in the final 3D coordinate space. The polygons we received were planes with many vertices (70 or more in some cases) listed in counter-clockwise order. Ultimately, the polygons needed to be broken into triangles for the renderer, but .obj supports polygons with arbitrary numbers of faces and Blender's .obj importer automatically converts large polygons to triangles, so we didn't need to address that problem directly. It is also worth noting that texture coordinates in the .obj format are floating point values between zero and one with the origin placed at the bottom-left corner of the image and all the data we received used pixel coordinates with the origin at the top-left corner, so we needed to perform a coordinate space transformation on the texture vertices. Apart from the texture coordinates, the format conversion was simply rearranging existing data into a format that Blender (and many other renderers) can read in.

After the files were converted, we took some steps toward accounting for errors in the model which included merging vertices and deleting erroneous polygons. In the initial data we received, vertices which obviously belonged together appeared separate from one another, due to minor errors and estimations made in previous stages of the pipeline. To account for these errors, we wrote an algorithm which merged similar vertices and faces.

To find like vertices, we simply calculated the distances between each pair of vertices and merged them if their distance was below a certain threshold. We set this threshold by experimenting with different values until we arrived upon a threshold that produced good results with our data set. To locate like faces, we first calculated a normal vector for each face and then chose faces with small angles between their normals. We then further reduced the set of faces by selecting faces with similar areas. Finally, we compared the axis-aligned bounding boxes of the faces. If all three of these differences were within

acceptable bounds, we concluded that the faces probably represented the same face in the physical world and we deleted the face with less detail.

3 Blender Modelling

After importing the data we need as an .obj file, Blender provides a method to view the data from the previous five teams. This is an important step in the pipeline, for the other teams cannot visualize their data without the help of this program. Giving input to the other teams based on what the model looks like is an important step in debugging. If the data is generally correct, a few minor corrections are made, then lighting is applied by hand. Tools in Blender make this easy to apply various types of lighting, depending on the nature of different light sources.

Normal maps and other related effects add a unprecedented and customizable level of realism to a mesh [8]. Unfortunately, we did not have enough data nor time to apply normal and specularity maps to the model based on texture. If we did have time, we would have used the Windows program xNormal to create normal maps for man-made surfaces [5]. By taking four photographs of what we want the texture of, with the light source in a different spot in each photo, and assigning it to relevant spots in the program, a tangent-spaced normal map is generated. The specularity map is then generated from the normal map with slight hand modification. We would then apply the maps by hand, using the UV mapping feature in Blender. This process would only have been repeated only a few times, since many of the building's textures (such as dry wall and whiteboard), are common throughout Regents.

4 Exporting Files

The initial export process is extremely straightforward. Irrlicht is a diverse engine that may import many formats. We felt the best format for our purpose was the COLLADA file format. This format saves textures, lighting, and other information in XML syntax, which was easily editable and correctable. However, COLLADA had difficulties exporting the various maps correctly. Therefore, baking the maps in Blender and exporting them as .png's would have been necessary if we had continued with our detailed modelling. New specularity maps and normal maps would have been created in this manner. These would then be applied separately in Irrlicht, using simple engine commands. Exporting shadow maps is not necessary for Irrlicht, since the engine automatically generates them based on the lighting.

One hassle in dealing with exporting meshes in Blender involves the direction of normals on the object. While generally, the normals are facing the way they should be, sometimes they are flipped, which Blender does not show in general preview mode. Textures are only one sided in Irrlicht and COLLADA, so care must be taken to correctly determine the destined direction in Blender.

Another challenge we faced was the limited polygon import in our version of Irrlicht. Due to the 16-bit indices in Irrlicht, only 65,535 vertices can be imported at one time. Therefore, each different .obj file we receive must be exported as a COLLADA file individually, and not combined with other .obj's. Because global coordinates remain the same in Irrlicht, the meshes always correctly align.

4 Other Work

Unfortunately, much of our work was hindered by a shortage of usable data from the teams charged with generating 3D polygons from source images. In response to this problem, we invested a significant portion of our time building tools that expedited and informed the other teams' debugging runs. Among these tools were a bad polygon finder, input file cleaners, and a script to distribute compute jobs on the campus clusters directly from lab workstations.

5 Conclusion

Our methods effectively created a model based on polygon data and coordinates. We first received the data and translated it into a .obj file, then merged related vertecies. In Blender, we corrected the models and would have applied various maps if we had the time. Finally, we exported the meshes into Irrlicht, which displayed the model we created. Due to the month-long time constraint, we did not have enough data from the other teams in order to effectively render the entirety of the Regents building. However, the entire rendering and display of a 3D Regents will eventually be completed. Our work will hopefully be used in the final creation of a 3D model of the Regents building.

Image Appendix

Figure 1: *Left stereo image of fish tank, taken by camera team.*

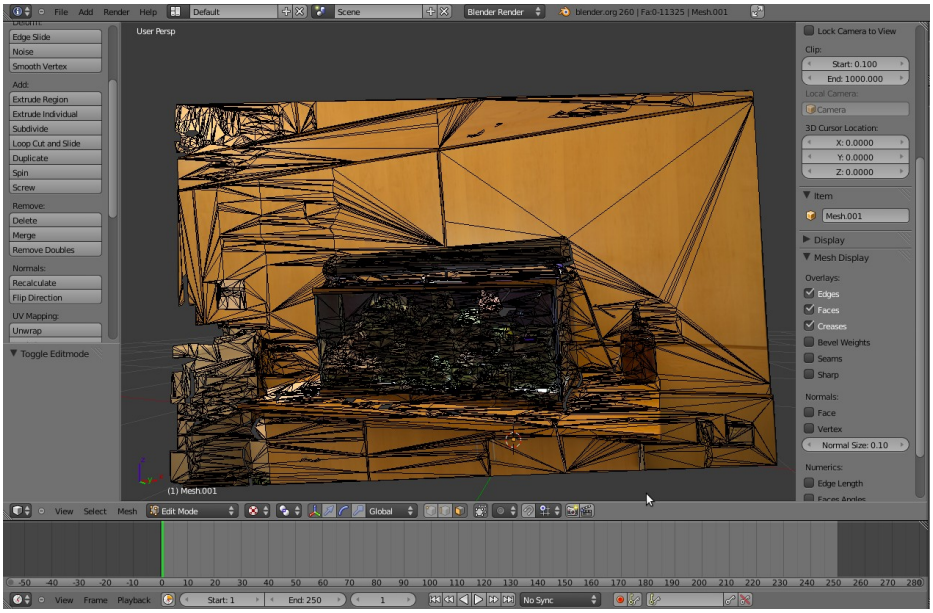

Figure 2: *Same view of a 3D model in the Blender edit interface, showing all polygons.*

References

[1] Agarwal, Sameer, Yasutaka Furukawa, et al. "Building Rome in a Day." *Communications of the ACM*. 54.10 (2011): 105-112. Web. 26 Jan. 2012. <http://grail.cs.washington.edu/rome/>.

[2] *Blender*. Blender Foundation, 14 Dec 2011. Web. 26 Jan 2012. <http://www.blender.org/>.

[3] Gebhardt, Nikolaus, et al. *Irrlicht Engine*. Nikolaus Gebhardt, 15 Nov 2010. Web. 26 Jan 2012. <http://irrlicht.sourceforge.net/>.

[4] "Getting support: the Blender community." *Blender Wiki*. Blender Foundation, 17 Jan 2012. Web. 26 Jan 2012. <http://wiki.blender.org/index.php/Doc:2.6/Manual/Introduction/Community>.

[5] Orgaz, Santiago. *xNormal*. Santiago Orgaz & co., 29 Dec 2011. Web. 26 Jan 2012. <http://www.xnormal.net/>.

[6] Salman, Nader, and Mariette Yvinec. "High resolution surface reconstruction from overlapping multiple-views." *Proceedings of the 25th annual symposium on Computational geometry (SCG* . (2009): n. page. Web. 26 Jan. 2012. <http://dl.acm.org/citation.cfm?id=1542362.1542386&coll=DL&dl=ACM>.

[7] Thormählen, Thorsten, and Hans-Peter Seidel. "3D-modelling by ortho-image generation from image sequences." *ACM Transactions on Graphics (TOG) - Proceedings of ACM* . 27.3 (2008): n. page. Web. 26 Jan. 2012. <http://dl.acm.org/citation.cfm?id=1399504.1360685&coll=DL&dl=ACM>.

[8] Toler-Franklin, Corey, Adam Finkelstein, and Szymon Rusinkiewicz. "Illustration of complex real-world objects using images with normals." *Proceedings of the 5th international symposium on Non-photorealistic animation and rendering (NPAR '07)*. (2007): n. page. Web. 26 Jan. 2012. <http://dl.acm.org/citation.cfm?id=1274871.1274889&coll=DL&dl=ACM>.

[9] Xiao, Jianxiong, Tian Fang, et al. "Image-based façade modelling." *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH Asia 2008*. 27.5 (2008): n. page. Web. 26 Jan. 2012. <http://dl.acm.org/citation.cfm?id=1457515.1409114&coll=DL&dl=ACM>.