RUTGERS, THE STATE UNIVERSITY OF NEW JERSEY

SENIOR CAPSTONE DESIGN, SPRING '13

Computer Vision-Based 3-D Reconstruction for Object Replication

Authors:
Ryan CULLINANE
Cady MOTYKA
Elie ROSEN

Advisor:
Professor Kristin DANA

April 28, 2013

Abstract

The Microsoft Kinect for Windows has proven to be a valuable tool in the field of computer vision. The Kinect is comprised of an infrared laser projector and depth sensor. The depth data of a scene is run through a bilateral filter and vector mathematics is used to define the coordinates, connecting lines, the vertices, and edges to form a three-dimensional mesh. The software displays the raw depth data and infrared camera image, this allows the user to filter out objects closer or further than a specified depth, and exports the reconstructed three-dimensional mesh. That mesh is then sliced into horizontal layers and converted into G-Code, a machine language that maneuvers the three-dimensional printer where to extrude the ABS plastic to create a physical replica of the reconstructed object.

Contents

1 Introduction

The Computer Vision-Based 3D Reconstruction for Object Replication is accomplished by using a Kinect for Windows. Originally, the Kinect was created for entertainment, but recently it has been introduced to the field of robotics and computer vision. The Kinect is a quick, reliable, and affordable tool that uses a near-infrared laser pattern projector and an IR camera, along with the sensor and software development kit, to calculate 3D measurements.

The 3D printer is another part of the robotics field that is beginning to find an increasing number of uses. The most innovative aspect of the 3D printer is the ability to print an object, regardless of interconnecting internal components, and have it function as intended. This means that any connecting gears that are printed with the 3D printer will in fact turn as they are supposed to.

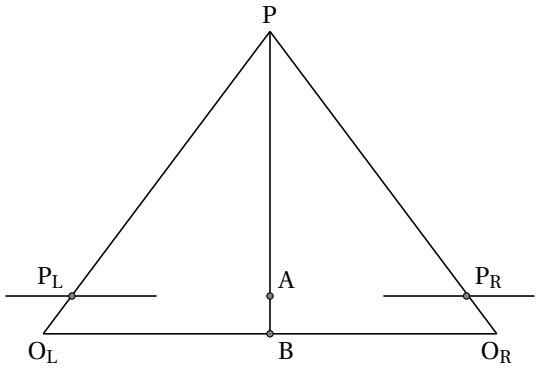
2 Methods

2.1 Calibration

The Kinect can be calibrated in a way similar to other cameras for computer vision, the only difference is that changes in the depth have to be present with the pattern in order to calibrate the depth camera. The Kinect needs to take an image of a checkerboard pattern.

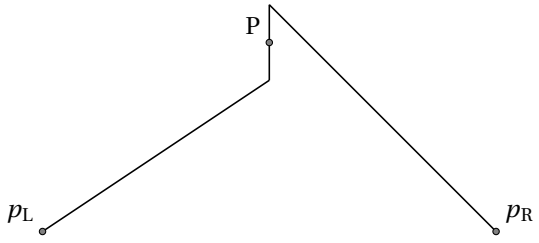
2.2 Stereo Reconstruction

Once the Kinect has been calibrated, all that is needed now for the stereo reconstruction is a triangulation of viewing rays.



P is the location of the object in the world, O_L and O_R are the left and right camera centers, P_R and P_L are the appearance of the point P in the two image planes where $P_L = \begin{bmatrix} x_L \\ y_L \end{bmatrix}$. The distance between O_L and O_R is T, or the distance between the left and right camera. The distance between A and B is the focal length of the cameras. If we define the distance between P and B as distance Z, the following equation can be used to represent the ratio between T and Z, using the theorem of like triangles: $\frac{T}{Z} = \frac{T+x_L-x_R}{Z-f} or \frac{T-x_R-x_L}{Z-f}$ Cross multiplying these equations results in: $\frac{Z(T-x_R-x_L)}{Z-f} = \frac{T(Z-f)}{Z}$ These calculations show that depth, or Z, is inversely proportional to disparity. This means that $P_L = \frac{f^L P}{Z_L}$. and $P_R = \frac{f^R P}{Z_R}$.

Once there is a corresponding point pair for P from the two images, an algorithm would undo the scale and shift of the pixel points in order to obtain the 2 dimensional camera coordinates. The midpoint algorithm is then used to find the real three-dimensional world coordinate that corresponds to that point pair.



Above are the rays $O_R\vec{p}_R$ and $O_L\vec{p}_L$ are drawn. The line connecting the two vectors, which is also perpendicular to both, is obtained by taking the cross product of these two vectors. The vector from p_L is equal to $a\vec{p}_L$, since point p_R is distance T away from p_L, the vector from p_R is equal to $b^L R_R \vec{p}_R + T$. The segment connecting these two

vectors can be represented as $c\vec{p}_{\mathrm{L}}x^{\mathrm{L}}\mathrm{R}_{\mathrm{R}}\vec{p}_{\mathrm{R}}$, where a, b and c are unknown constants that can be solved using the three equations explained above.

The point P lies on the center of this line and be found by ${}^{\mathrm{L}}\mathrm{P} = a\vec{p}_{\mathrm{L}} + \frac{c}{2}\vec{p}_{\mathrm{L}}x^{\mathrm{L}}\mathrm{R}_{\mathrm{R}}\vec{p}_{\mathrm{R}}$ In order to get the world point M, this point would just be divided by the Intrinsic and extrinsic matrices.

KINECT
for XBOX 360.

Figure 1: The Xbox Kinect, the sensor on the left is the infrared light source, the center is a RGB camera and the 3D depth camera is on the right. in addition to these cameras, the base has a motorized tilt and a multi-array mic goes along the bottom of the wand.

The Kinect accomplishes this triangulation by using the known information about the sensor, the data obtained from the infrared projection and the image received from the camera. The sensor will project invisible light onto an object, the light bounces back and the infrared sensor reads back the data. These clusters of light that are read back can be matched to the hard-coded images the Kinect has of the normal projected pattern and allows for a search for correlations, or the matching points. While looking through the camera's focal point, the point of interest will fall on a specific pixel, depending on how close or far away it is, this means that we know along which trajectory this point is from the camera. The relative line of trajectory from the projector and from the camera, along with the known information about the distance between the cameras on the Kinect sensor, are used in the above described triangulation process to find the three-dimensional coordinates of the point. Figure 1 shows how the three cameras are arranged on the Kinect.

2.3 Bilateral Filter and Image Pyramiding

In order to make this data more manageable, a bilateral filter is used to remove the erroneous measurements. This filter will just take every point, and recalculate the value of that point based on the waited average of the surrounding pixels in a specified neighborhood. The process takes away some of the sharpness of the depth map, but it removes the noise that will skew the results of the three dimensional reconstruction. The filter takes every pixel in the image and replaces its value with $\mathrm{BF[I]}_{\mathrm{P}} = \frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$ times the neighborhood of the 3 by 3 square of pixels around the pixel that is being changed. This resulting pixel value represents the average of the 9 pixels, where the closer pixels weight in the average is heavier than the further pixel values. Next, in order to represent the depth map in the true three dimensional points, a vertex must be created at each point where x, and y are the pixel values and the depth is the z coordinate. These points are then multiplied by a calibrated matrix to convert them into a vector map, or point cloud. The normal of each vertex is calculated by the cross product of the neighboring pixels. This process is combined with the process of computing a pyramid of the data, multiple copied of the depth map are made with smaller resolutions, at leach layer the vertices and their normals are calculated and stored.

2.4 Vector Field and Truncated Surface Distance Function

The next step is to take a previously calculated vertex and normal map and run an Iterative Closest Point Algorithm on the four maps. This algorithm associates points by the nearest neighbor criteria, estimates a transformation using a mean square cost function, transform the point using the estimated parameter and then re-associates the points iteratively. This step creates a final rotation and translation that minimizes distant errors between the point clouds. This algorithm is useful because it will find the best way to position the point cloud before the reconstruction so that every part of the object is represented correctly. Once the best fit is found, the existing depth data can be combined with the existing model to get a more refined result. The filtering got rid of the noise, and adding the raw depth data back to this will add the important details back into the final model. A Truncated Surface Distance Function is used to fit the depth data and the model back together. Finally, a weighted average of the existing model and the latest depth measurements is taken and used to generate the surface using a ray-casting algorithm, to express the reconstructed object to the user. The model can be exported as an STL file to the Gcode converter, but some work still needs to be done in order to represent to the user what he ore she is trying to print. The ray-casting algorithm will tell a virtual camera looking at the virtual model what to display on the GUI. A ray is cast from every pixel in the image through the focal point of the digital camera, and the first surface that the ray intersects with is excited, displaying it to the user. [1]

2.5 G-code Conversion

The RepRap firmware uses G-code to communicate to the 3D printer, specifically to define the print head movements. This code has commands that tell the print head to move to a certain point with rapid or controlled movement, turn on a cooling fan, or selecting a different tool. Since this 3D printer does not have as many features, the G-code generator does not have to add much complicated code, but rather just instructions to the printer head. Since the printer continuously dispenses plastic, it is necessary to find a path for it to take that will build up the reconstructed object layer by layer without placing too much plastic in any specific area. This requires cutting up the reconstructed object into layers and then finding the best path to traverse that layer without overlapping any part of that path. The G-code converter takes in the STL file, cuts it up into horizontal layers and then calculates the about of material that is needed to fill each slice.

3 Experimental Results

Figure 2: The Kinect raw depth field, with the RGB image mapped to it, without bilateral filter

Figure 3: Triangles representing the 3D data

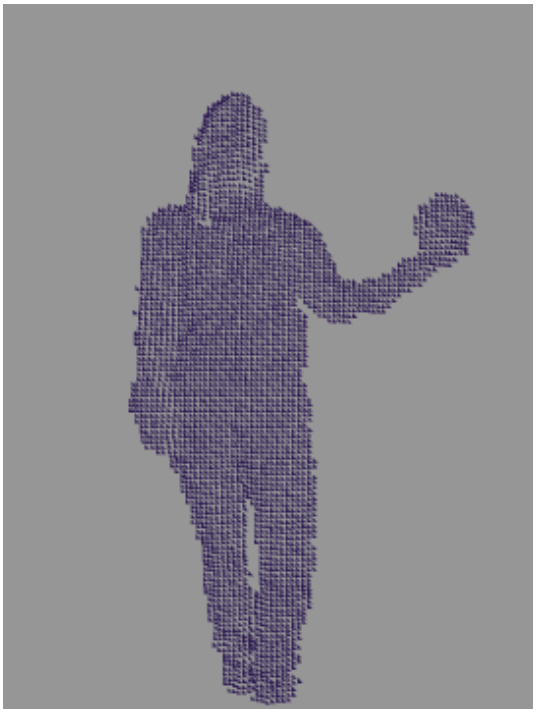


Figure 4: The 3D data with the correct background filtered out

Figure 5: Comparison of the raw depth data and the Triangles that represent the 3D data

4 Discussion

Part of the issue of working with this 3D printer was that many of the parts used in the construction of the machine were actually printed by another 3D printer. This meant that before we could start the construction of the printer, we had to wait on all of the correct parts to be printed. Even once we thought that we had all of the required pieces, we found two important parts that we were missing that we have to go to the Rutgers Maker Space to get printed. We also found that many of the plastic parts were made to be the exact size of the rods that needed to be placed into them, this meant that it took a lot of force to get some of the components to fit together properly. At one point we tried to use hot water to make the printed plastic more malleable but we fear that this made some of the pieces warp.

Another big issues that was found with he construction of the 3D printer was the lack of good document ion on assembly. For the triangular base structure had to be taken apart and reassembles multiple time sin order to fix errors. One example was the motor bracket for the motor that controls the movement along the y-axis, there was not good diagrams to show which side of the machine this part should be placed and what direction it should be oriented. Figure 6 shows a number of the unlabeled parts that we received from IEEE. Since documentation as hard to find and none of the parts were labeled, we also had trouble finding the correct STL files to send in in order to get ore parts. We knew that we were missing the brackets that connect the two top motors to the threaded rod that controls the movement along the z-axis, but we did not know what file needed to be printed.

Figure 6: Parts of the 3D Printer

Even once the mechanical parts of the 3D printer have been constructed, we still need to calibrate all of the axes, add all of the electrical components, calibrate the firmware and build the protective frame around the printer. Figure 7 shows the constructed frame, x,y and z-axes and the installed printed. One of the issues that we will run

into as we complete the 3D printer is the extruder. This component is responsible for heating and melting the ABS plastic and placing it onto the right place on the heat bed. We have discovered a problem that normal solder cannot handle the heat that is needed to melt the ABS plastic and the piece falls apart as the machine heats up. In order to fix this issue, we need to order silver solder that will be capable of withstanding a temperature of 221 degrees Fahrenheit.

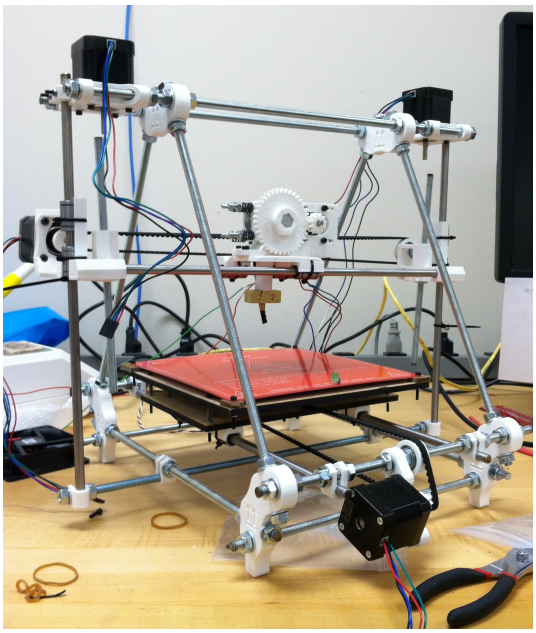


Figure 7: Construction of the 3D Printer Frame and x,y and z-axes

The biggest issue that we have had with the software component of this project was the lack of examples. Other people who have worked on similar projects had been using the original version of the Microsoft Kinect Software Development Kit. This means that a many implementations used packages that are no longer part of the SDK that we have to work with. We've had to find ways to make the new SDK, which was released in 2012, work in a way similar to the old SDK.

5 Cost Analysis

6 Current Trends in Robotics and Computer Vision

6.1 Kinect Revolution

One of the reasons that the Kinect has become so popular for computer vision projects is that it is a cheap, quick, and highly reliable for 3D measurements. Many researchers are beginning to look into the possibility of using this device to achieve everything from a 3D reconstruction of a scene to aiding in a SLAM algorithm. The fact that this device is so affordable, and so many new resources are available, makes the Kinect a viable device for conducting research in the field of robotics and computer vision.

The KinectFusion Project is slightly different than other projects that were using the Kinect; instead of using both the RGB cameras and the sensor, this project tracks the 3D sensor pose and preforms a reconstruction in real time using exclusively the depth data. This paper points out that depth cameras aren't exactly new, but the

Kinect is a low-cost, real-time, depth camera that is much more accessible. The accuracy of the Kinect is called into questions, the point cloud that the depth data creates does usually contain noise and sometimes has holes where no readings were obtained. This project also considered the Kinect's low X/Y resolution and depth accuracy and fixes the quality of the images using depth super resolution. KinectFusion also looks into using multiple Kinects to preform a 3D body scan; this raises more issues because the quality of the overlapping sections of the images is compromised.

Another KinectFusion Project is the Real-time 3D Reconstruction and Interaction, this project is impressive because the entire process is done using a moving depth camera. With this software, the user can hold a Kinect camera up to a scene, and a 3D construction would be made. Not only would the user be able to see the 3D Reconstruction, but they would be able to interact with it; for instance, if they were to throw a handful of spheres onto the scene, they would land on the top of appropriate surfaces, and fall under appropriate objects following the rules of physics. To accomplish this, the depth camera is used to track the 3D pose and the sensor is used to reconstruct the scene. Different views of the scene are taken and fussed together into a singe representation, the pipe line segments the objects in the scene and uses them to create a global surface based reconstruction. This project shows the real-time capabilities of then Kinect and why that makes it an innovative tool for computer vision

A study shown in the Asia Simulation Conference in 2011 demonstrated that a calibrated Kinect can be combined with Structure from Motion to find the 3D data of a scene and reconstruct the surface by Multi-view Stereo. This study proved that the Kinect was more accurate for this procedure than a SwissRanger SR-4000 3D-TOF camera and close to a medium resolution SLR Stereo rigs. The Kinect works by using a near-infrared laser pattern projector and an IR camera as a stereo pair to triangulate points in 3D space, then the RGB camera is used to reconstruct the correct texture to the 3D points. This RGB camera, which outputs medium quality images, can also be used for recognition. One issue this study found was that the resulting IR and Depth images were shifted. To figure out what the shift was, the Kinect recorded pictures of a circle from different distances. The shift was found to be around 4 pixels in the u direction and three pixels in the v direction. Even after the camera has been fully calibrated, there are a few remaining residual errors in the close range 3D measurements. An easy fix for this error was to we form a z-correction image of z values constructed as the pixel-wise mean of all residual images and then subtract that correction image from the z coordinates of the 3D image.[2] Though the SLR Stereo was the most accurate, the error e (or the Euclidean distance between the points returned by the sensors and points reconstructed in the process of calibration) of the SR-400 was much higher than the Kinect and the SLR. This study shows that the Kinect is possible cheaper and simpler alternative to previously used cameras and rigs in the computer vision field.

Another subject of research that is looking into using the Kinect is the simultaneous localization and mapping algorithm, used to create a 3D map of the world so that the robot can avoid collision with obstacles or walls. The SLAM problem could be solved using GPS if the robot is outside, but inside one needs to use wheel or visual odometry. Visual odometry determines the position and the orientation of the robot using the associated camera images, algorithms like Scale Invariant Feature Transformation (SIFT), used to find the interest points, and laser sensors, used to collect depth data. Since the Kinect has both the RGB camera and a laser sensor, this piece of technology is a good piece of hardware to use for robots computing the SLAM Algorithm. In the study conducted by the students in the Graduates School of Science and Technology, at Meiji University, they found that the Kinect worked well for this process for horizontal and straight movement, but they had errors when they tried to recreate an earlier experiment, this means that their algorithm successfully solves the initial problem, but accuracy fell over time.[5] They found that the issue was not with the Kinect, and that it could be solved using the Speed-Up Robust Feature algorithm (SURF) and Smirnov-Grubbs test to further improve the accuracy of their SLAM Algorithm. This study proved that the Kinect was a reasonable, inexpensive and non-special piece of equipment that is capable of preforming

well in computer vision applications.

It seems as though the Kinect is a popular choice in current robotics and computer vision. This device is affordable, easily obtainable, and capable of a lot more than is expected from a video game add on. The Kinect combines a near-infrared laser pattern projector and an IR camera in one tool, and when combined with this eliminates the set up of some other configuration. The Kinect is also surprisingly accurate, requiring only some optimization software to make the results comparable to the results from a medium resolution SLR Stereo rig.

6.2 3-D Printing Future

One of the most innovated uses for the 3D Printer is its applications in the medical field. Since 2010, people have been using 3D printers to print out prosthetic limbs. One company in California has been printing the totally customizable prosthetics, which cost about one tenth of traditional prothetic limbs. Another company is looking at the possibility of using a 3D printer to print a house. Right now the design fits on the back of a tractor trailer and the 3D printer prints out custom concrete parts the are then assembled to complete the house. Some 3D printers have the ability to change the printing head, so it can begin printing with one material and then switch to a different material, all based on the code it receives, this means that a 3D printer could theoretically print the concrete part of the house and switch to printing the plastic siding or the glass windows all on the same path around the outside of the house. The most importunes aspect of these 3D printer applications is that it drastically cuts down on production costs, allowing the consumer to pay a lower price and get a completely customized product. Rather than paying a person to design the object, and then have a bother construct it, with a 3D printer all that needs to be done is the design and the 3D printer automates the entire construction process. For example, the 3D printed prosthetics cost 5,000 dollars to print and customize by covering the 3D printed material in a shoe or sleeve while a normal generic prothetic would cost about 60,000 dollars.[6] The 3D printer is a piece of technology that could continue to make the price of consumer goods fall and allow for more customization than has ever existed for consumer products.

In recent news, biomedical scientists have taken the 3D printing technology a step further than prosthetics. A man had 75% of his skull replaced but a 3D printed implant made by Oxford Performance Materials. Plastics have been used since the 1940's to replace missing bone fragments, and 3D modeling techniques have been used to match the size and shape of the plastic to someones skull. This Connecticut based company combined the 3D modeling techniques with 3D printing technology to produce the replacement part that only took five days to fabricate.[**?**] The material that it is printed with has some of the same properties as bones and are osteoconductive, so the skull will actually grow and attach itself to the implant. This is also much better than metals that would block doctors from seeing past the implant in X-rays. This company is now also looking at using this procedure to 3D print other replacement bones for victims or cancerous bone or trauma.

Even though lower cost of production is a goal for many industries, the 3D printing technology can be considered a disruptive technology, meaning that over the course of a short period of time it could change an existing market and value network while replacing existing technology. An article in the Harvard Business Review explains that good would be produced at or close to their point of purchase or consumption. Even if this ins't the case with every industry, the cost will be offset by the elimination of shipping the completed object to the consumer, something like car parts could be printed in a metropolitan area rather than made and shipped from a factory. This article also mentions how the 3D printer would allow for cheap and efficient customization of these products. Since changing the shape, color or material of what you want to print is only a mater of changing code, the first model could be relatively different than the second model for virtually no extra cost. [3] The 3D printer could potentially affect the global market. Many products are manufactured overseas since it is much cheaper for the pieces to be created an assembled by underpaid workers, when 3D printing is perfected the parts could be made and assembled

by a machine in the US for less than it cost to have the product manufactured and shipped from overseas.

An article in Machine Design talks about what changes are being made to the 3D printers in order to make them more durable, user friendly and affordable. One brand, LeapFrog, has made the entire device out of aluminum and replaced the stepper-motor drivers with professional drivers that last longer. This company has also added a duel option extruder so that the printer could construct something like a bridge by adding the plastic from one extruder and a water soluble support system with the other extrude that can be washed away once the printing is finished. This printer also uses PLA plastic, which is more brittle and has a lower melting temperature, can print smoother edges than the usual ABS plastic. Another brand, FormLabs, uses a liquid photopolymer instead of a spool of plastic, this resin cuts the price of printing materials in half and allows for a layer thickness of only 25 microns. The RepRap 3D printer has been designated a self-replicating printer because it can be used to print parts for constructing another 3D printer. It is believed that between 20,000 and 30,000 of these machines are now in existence.[4] The company Staples has started "Staples Easy 3D" in Belgium and the Netherlands, where anyone can upload their file to the center and later pick up the 3D model at their local staples or have it shipped to their house. Services like this are a sign that 3D printing will soon be as mainstream as 2D printing.

7 Appendix

0.1 KinectScan Application Code

```
namespace kinectScan
{
    using System;
    using System.ComponentModel;
    using System.Globalization;
    using System.IO;
    using System.Threading.Tasks;
    using System.Drawing;
    using System.Diagnostics;
    using System.Windows;
    using System.Windows.Controls;
    using System.Windows.Media;
    using System.Windows.Media.Imaging;
    using System.Windows.Media.Media3D;
    using System.Windows.Threading;

    using HelixToolkit.Wpf;

    using Microsoft.Kinect;
    using Microsoft.Kinect.Toolkit;

    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {

        /// <summary>
        /// Timestamp of last depth frame in milliseconds
        /// </summary>
        private long lastFrameTimestamp = 0;

        /// <summary>
        /// Timer to count FPS
        /// </summary>
        private DispatcherTimer fpsTimer;

        /// <summary>
        /// Timer stamp of last computation of FPS
        /// </summary>
        private DateTime lastFPSTimestamp;

        /// <summary>
        /// Event interval for FPS timer
        /// </summary>
        private const int FpsInterval = 5;

        /// <summary>
        /// The counter for frames that have been processed
        /// </summary>
        private int processedFrameCount = 0;

        /// <summary>
        /// Active Kinect sensor
        /// </summary>
        private KinectSensor sensor;

        /// <summary>
        /// Kinect sensor chooser object
        /// </summary>
        private KinectSensorChooser sensorChooser;

        /// <summary>
        /// Format of depth image to use
        /// </summary>
        private const DepthImageFormat dFormat = DepthImageFormat.Resolution320x240Fps30;

        /// <summary>
```

```
69          /// Format of color image to use
70          /// </summary>
71          private const ColorImageFormat cFormat = ColorImageFormat.InfraredResolution640x480Fps30;
72
73          // stores furthest depth in the scene
74          public ushort greatestDepth = 0;
75
76          // array for all of the depth data
77          private int[] Depth = new int[320 * 240];
78
79          // stores all of the 3D trianlges with normals and points
80          Model3DGroup modelGroup = new Model3DGroup();
81
82          // material placed over the mesh for viewing
83          public GeometryModel3D msheet = new GeometryModel3D();
84
85          // collection of corners for the triangles
86          public Point3DCollection corners = new Point3DCollection();
87
88          // collection of all the triangles
89          public Int32Collection Triangles = new Int32Collection();
90
91
92          public MeshGeometry3D tmesh = new MeshGeometry3D();
93
94          // collection of all the cross product normals
95          public Vector3DCollection Normals = new Vector3DCollection();
96
97          // add texture to the mesh
98          public PointCollection myTextureCoordinatesCollection = new PointCollection();
99
100         // storage for camera, scene, etc...
101         public ModelVisual3D modelsVisual = new ModelVisual3D();
102
103
104         public Viewport3D myViewport = new Viewport3D();
105
106         // test variable
107         public int samplespot;
108
109         // variable for changing the quality 1 is the best 16 contains almost no data
110         public int s = 1;
111
112         // depth point collection
113         public int[] depths_array = new int[4];
114
115         // collection of points
116         Point3D[] points_array = new Point3D[4];
117
118         // collection of vectors
119         Vector3D[] vectors_array = new Vector3D[5];
120
121         //used for displaying RGB camera
122         public byte[] colorPixels;
123         public WriteableBitmap colorBitmap;
124
125         public MainWindow()
126         {
127             InitializeComponent();
128         }
129
130         private void WindowLoaded(object sender, RoutedEventArgs e)
131         {
132             // Start Kinect sensor chooser
133             this.sensorChooser = new KinectSensorChooser();
134             this.sensorChooserUI.KinectSensorChooser = this.sensorChooser;
135             this.sensorChooser.KinectChanged += this.OnKinectSensorChanged;
136             this.sensorChooser.Start();
```

```
137
138             // Start fps timer
139             this.fpsTimer = new DispatcherTimer(DispatcherPriority.Send);
140             this.fpsTimer.Interval = new TimeSpan(0, 0, FpsInterval);
141             this.fpsTimer.Tick += this.FpsTimerTick;
142             this.fpsTimer.Start();
143
144             // Set last fps timestamp as now
145             this.lastFPSTimestamp = DateTime.Now;
146         }
147
148         /// <summary>
149         /// Execute shutdown tasks
150         /// </summary>
151         /// <param name="sender">object sending the event</param>
152         /// <param name="e">event arguments</param>
153         private void WindowClosing(object sender, System.ComponentModel.CancelEventArgs e)
154         {
155             // Stop timer
156             if (null != this.fpsTimer)
157             {
158                 this.fpsTimer.Stop();
159                 this.fpsTimer.Tick -= this.FpsTimerTick;
160             }
161
162             // Unregister Kinect sensor chooser event
163             if (null != this.sensorChooser)
164             {
165                 this.sensorChooser.KinectChanged -= this.OnKinectSensorChanged;
166             }
167
168             // Stop sensor
169             if (null != this.sensor)
170             {
171                 this.sensor.Stop();
172                 this.sensor.DepthFrameReady -= this.SensorDepthFrameReady;
173                 this.sensor.ColorFrameReady -= this.SensorColorFrameReady;
174             }
175
176             // Empty the canvas
177             this.ClearMesh();
178         }
179
180         /// <summary>
181         /// Handles adding a new kinect
182         /// </summary>
183         /// <param name="sender">object sending the event</param>
184         /// <param name="e">event arguments for the newly connected Kinect</param>
185         private void OnKinectSensorChanged(object sender, KinectChangedEventArgs e)
186         {
187             // Check new sensor's status
188             if (this.sensor != e.NewSensor)
189             {
190                 // Stop old sensor
191                 if (null != this.sensor)
192                 {
193                     this.sensor.Stop();
194                     this.sensor.DepthFrameReady -= this.SensorDepthFrameReady;
195                     this.sensor.ColorFrameReady -= this.SensorColorFrameReady;
196                 }
197
198                 this.sensor = null;
199
200                 if (null != e.NewSensor && KinectStatus.Connected == e.NewSensor.Status)
201                 {
202                     // Start new sensor
203                     this.sensor = e.NewSensor;
204                     this.StartCameraStream(dFormat, cFormat);
```

```
205                 }
206             }
207 
208             if (null == this.sensor)
209             {
210                 // if no kinect clear the text on screen
211                 this.statusBarText.Content = Properties.Resources.NoKinectReady;
212                 this.IR_Title.Content = "";
213                 this.Model_Title.Content = "";
214                 this.RGB_Title.Content = "";
215             }
216         }
217 
218         /// <summary>
219         /// Handler for FPS timer tick
220         /// </summary>
221         /// <param name="sender">Object sending the event</param>
222         /// <param name="e">Event arguments</param>
223         private void FpsTimerTick(object sender, EventArgs e)
224         {
225 
226             if (null == this.sensor)
227             {
228                 // Show "No ready Kinect found!" on status bar
229                 this.KinectStatusText.Content = Properties.Resources.NoReadyKinect;
230             }
231             else
232             {
233                 // Calculate time span from last calculation of FPS
234                 double intervalSeconds = (DateTime.Now - this.lastFPSTimestamp).TotalSeconds;
235 
236                 // Calculate and show fps on status bar
237                 this.KinectStatusText.Content = string.Format(
238                     System.Globalization.CultureInfo.InvariantCulture,
239                     Properties.Resources.Fps,
240                     (double)this.processedFrameCount / intervalSeconds);
241             }
242 
243             // Reset frame counter
244             this.processedFrameCount = 0;
245             this.lastFPSTimestamp = DateTime.Now;
246         }
247 
248         /// <summary>
249         /// Reset FPS timer and counter
250         /// </summary>
251         private void ResetFps()
252         {
253             // Restart fps timer
254             if (null != this.fpsTimer)
255             {
256                 this.fpsTimer.Stop();
257                 this.fpsTimer.Start();
258             }
259 
260             // Reset frame counter
261             this.processedFrameCount = 0;
262             this.lastFPSTimestamp = DateTime.Now;
263         }
264 
265         /// <summary>
266         /// Start depth stream at specific resolution
267         /// </summary>
268         /// <param name="format">The resolution of image in depth stream</param>
269         private void StartCameraStream(DepthImageFormat dFormat, ColorImageFormat cFormat)
270         {
271             try
272             {
```

```
273                 // Enable streams, register event handler and start
274                 this.sensor.DepthStream.Enable(dFormat);
275                 this.sensor.DepthFrameReady += this.SensorDepthFrameReady;
276                 this.sensor.ColorStream.Enable(cFormat);
277                 this.sensor.ColorFrameReady += this.SensorColorFrameReady;
278                 this.sensor.Start();
279             }
280             catch (IOException ex)
281             {
282                 // Device is in use
283                 this.sensor = null;
284                 this.ShowStatusMessage(ex.Message);
285 
286                 return;
287             }
288             catch (InvalidOperationException ex)
289             {
290                 // Device is not valid, not supported or hardware feature unavailable
291                 this.sensor = null;
292                 this.ShowStatusMessage(ex.Message);
293 
294                 return;
295             }
296 
297             // Allocate space to put the pixels we'll receive
298             this.colorPixels = new byte[this.sensor.ColorStream.FramePixelDataLength];
299 
300             //// This is the bitmap we'll display on-screen
301             this.colorBitmap = new WriteableBitmap(this.sensor.ColorStream.FrameWidth, this.sensor.
    ColorStream.FrameHeight, 96.0, 96.0, PixelFormats.Gray16, null);
302         }
303 
304         /// <summary>
305         /// Event handler for Kinect sensor's ColorFrameReady event
306         /// </summary>
307         /// <param name="sender">object sending the event</param>
308         /// <param name="e">event arguments</param>
309         void SensorColorFrameReady(object sender, ColorImageFrameReadyEventArgs e)
310         {
311             using (ColorImageFrame colorFrame = e.OpenColorImageFrame())
312             {
313                 if (colorFrame != null)
314                 {
315                     // Copy the pixel data from the image to a temporary array
316                     colorFrame.CopyPixelDataTo(this.colorPixels);
317 
318                     // Write the pixel data into our bitmap
319                     this.colorBitmap.WritePixels(
320                         new Int32Rect(0, 0, this.colorBitmap.PixelWidth, this.colorBitmap.PixelHeight),
321                         this.colorPixels,
322                         this.colorBitmap.PixelWidth * colorFrame.BytesPerPixel,
323                         0);
324                 }
325 
326                 // set the RGB image to the RGB camera
327                 this.KinectRGBView.Source = this.colorBitmap;
328 
329             }
330         }
331 
332         /// <summary>
333         /// Event handler for Kinect sensor's DepthFrameReady event
334         /// Take in depth data
335         /// </summary>
336         /// <param name="sender">object sending the event</param>
337         /// <param name="e">event arguments</param>
338         void SensorDepthFrameReady(object sender, DepthImageFrameReadyEventArgs e)
339         {
```

```
340
341             DepthImageFrame imageFrame = e.OpenDepthImageFrame();
342             if (imageFrame != null)
343             {
344                 double maxDepth = Far_Filter_Slider.Value;
345                 short[] pixelData = new short[imageFrame.PixelDataLength];
346                 imageFrame.CopyPixelDataTo(pixelData);
347                 this.greatestDepth = 0;
348                 for (int y = 0; y < 240; y++)
349                 {
350                     for (int x = 0; x < 320; x++)
351                     {
352                         // scale depth down
353                         this.Depth[x + (y * 320)] = ((ushort)pixelData[x + y * 320]) / 100;
354 
355                         // finds the furthest depth from all the depth pixels
356                         if ((this.Depth[x + y * 320] > this.greatestDepth) && (this.Depth[x + y * 320] ↙
    < maxDepth))
357                         {
358                             this.greatestDepth = (ushort)this.Depth[x + y * 320];
359                         }
360 
361 
362                     }
363                 }
364                 // Blur Filter  -- Guassian
365                 if (Filter_Blur.IsChecked == true)
366                 {
367                     for (int i = 641; i < this.Depth.Length - 641; ++i)
368                     {
369 
370                         short depthaverage = (Int16)((this.Depth[i - 641] + (2 * this.Depth[i - 640]) + ↙
     this.Depth[i - 639] +
371                                                     (2 * this.Depth[i - 1]) + (4 * this.Depth[i]) + (2 ↙
     * this.Depth[i + 2]) +
372                                                     this.Depth[i + 639] + (2 * this.Depth[i + 640]) + ↙
    this.Depth[i + 641]) / 16);
373 
374                         this.Depth[i] = depthaverage;
375                         if ((this.Depth[i] > this.greatestDepth) && (this.Depth[i] < maxDepth))
376                         {
377                             this.greatestDepth = (ushort)this.Depth[i];
378                         }
379                     }
380                 }
381 
382                 // Set the depth image to the Depth sensor view
383                 this.KinectDepthView.Source = DepthToBitmapSource(imageFrame);
384             }
385         }
386 
387 
388         /// <summary>
389         /// Flag check for a point within the bounding box
390         /// </summary>
391         /// <param name="x">location on the x plane</param>
392         /// <param name="y">location on the y plane</param>
393         private bool PointinRange(int x, int y)
394         {
395             double minDepth = Near_Filter_Slider.Value;
396             double maxDepth = Far_Filter_Slider.Value;
397             return ((this.Depth[x + (y * 320)] >= minDepth && this.Depth[x + (y * 320)] <= maxDepth) ||
398                 (this.Depth[(x + s) + (y * 320)] >= minDepth && this.Depth[(x + s) + (y * 320)] <=      ↙
    maxDepth) ||
399                 (this.Depth[x + ((y + s) * 320)] >= minDepth && this.Depth[x + ((y + s) * 320)] <=      ↙
    maxDepth) ||
400                 (this.Depth[(x + s) + ((y + s) * 320)] >= minDepth && this.Depth[(x + s) + ((y + s) *   ↙
    320)] <= maxDepth));
```

```
401
402         }
403
404         /// <summary>
405         /// Create the mesh
406         /// </summary>
407         void BuildMesh()
408         {
409             double maxDepth = Far_Filter_Slider.Value;
410             int i = 0;
411             for (int y = (int)Top_Slider.Value; y < ((int)Bot_Slider.Value - s); y = y + s)
412             {
413                 for (int x = (int)Left_Slider.Value; x < ((int)Right_Slider.Value - s); x = x + s)
414                 {
415                     //Any point less than max
416                     if (PointinRange(x, y))
417                     {
418                         if (this.Depth[x + ((y + s) * 320)] >= maxDepth)
419                         {
420                             depths_array[0] = -this.greatestDepth;
421                         }
422                         else
423                         {
424                             depths_array[0] = -this.Depth[x + ((y + s) * 320)];
425                         }
426
427                         if (this.Depth[x + (y * 320)] >= maxDepth)
428                         {
429                             depths_array[1] = -this.greatestDepth;
430                         }
431                         else
432                         {
433                             depths_array[1] = -this.Depth[x + (y * 320)];
434                         }
435
436                         if (this.Depth[(x + s) + (y * 320)] >= maxDepth)
437                         {
438                             depths_array[2] = -this.greatestDepth;
439                         }
440                         else
441                         {
442                             depths_array[2] = -this.Depth[(x + s) + (y * 320)];
443                         }
444
445                         if (this.Depth[(x + s) + ((y + s) * 320)] >= maxDepth)
446                         {
447                             depths_array[3] = -this.greatestDepth;
448                         }
449                         else
450                         {
451                             depths_array[3] = -this.Depth[(x + s) + ((y + s) * 320)];
452                         }
453
454                         // triangle point locations
455                         points_array[0] = new Point3D(x, (y + s), depths_array[0]);
456                         points_array[1] = new Point3D(x, y, depths_array[1]);
457                         points_array[2] = new Point3D((x + s), y, depths_array[2]);
458                         points_array[3] = new Point3D((x + s), (y + s), depths_array[3]);
459
460                         // create vectors of size difference between points
461                         vectors_array[0] = new Vector3D(points_array[1].X - points_array[0].X,
    points_array[1].Y - points_array[0].Y, points_array[1].Z - points_array[0].Z);
462                         vectors_array[1] = new Vector3D(points_array[1].X - points_array[2].X,
    points_array[1].Y - points_array[2].Y, points_array[1].Z - points_array[2].Z);
463                         vectors_array[2] = new Vector3D(points_array[2].X - points_array[0].X,
    points_array[2].Y - points_array[0].Y, points_array[2].Z - points_array[0].Z);
464                         vectors_array[3] = new Vector3D(points_array[3].X - points_array[0].X,
    points_array[3].Y - points_array[0].Y, points_array[3].Z - points_array[0].Z);
```

```
                        vectors_array[4] = new Vector3D(points_array[2].X - points_array[3].X,
    points_array[2].Y - points_array[3].Y, points_array[2].Z - points_array[3].Z);

                        // add the corners to the 2 triangles to form a square
                        corners.Add(points_array[0]);
                        corners.Add(points_array[1]);
                        corners.Add(points_array[2]);
                        corners.Add(points_array[2]);
                        corners.Add(points_array[3]);
                        corners.Add(points_array[0]);

                        // add triangles to the collection
                        Triangles.Add(i);
                        Triangles.Add(i + 1);
                        Triangles.Add(i + 2);
                        Triangles.Add(i + 3);
                        Triangles.Add(i + 4);
                        Triangles.Add(i + 5);

                        // find the normals of the triangles by taking the cross product
                        Normals.Add(Vector3D.CrossProduct(vectors_array[0], vectors_array[2]));
                        Normals.Add(Vector3D.CrossProduct(vectors_array[0], vectors_array[1]));
                        Normals.Add(Vector3D.CrossProduct(vectors_array[1], vectors_array[2]));
                        Normals.Add(Vector3D.CrossProduct(vectors_array[1], vectors_array[2]));
                        Normals.Add(Vector3D.CrossProduct(vectors_array[3], vectors_array[4]));
                        Normals.Add(Vector3D.CrossProduct(vectors_array[0], vectors_array[2]));

                        i = i + 6;
                    }

                }
            }

            // add the flat back wall
            int numcorners = corners.Count;
            for (int p = 0; p < numcorners; p++)
            {
                Point3D cornertocopy = corners[p];
                corners.Add(new Point3D(cornertocopy.X, cornertocopy.Y, -this.greatestDepth));
                Triangles.Add(i);
                Normals.Add(new Vector3D(0, 0, 1));
                i = i + 1;
            }


        }

        /// <summary>
        /// Create depth image from depth frame
        /// </summary>
        /// <param name="imageFrame">collection of depth data</param>
        BitmapSource DepthToBitmapSource(DepthImageFrame imageFrame)
        {
            short[] pixelData = new short[imageFrame.PixelDataLength];
            imageFrame.CopyPixelDataTo(pixelData);
            BitmapSource bmap = BitmapSource.Create(
             imageFrame.Width,
             imageFrame.Height,
             96, 96,
             PixelFormats.Gray16,
             null,
             pixelData,
             imageFrame.Width * imageFrame.BytesPerPixel);
            return bmap;
        }

        /// <summary>
        /// take a photo when button is clicked
```

```
532         /// </summary>
533         /// <param name="sender">object sending the event</param>
534         /// <param name="e">event arguments</param>
535         private void Begin_Scan_Click(object sender, RoutedEventArgs e)
536         {
537             //clear the canvas
538             this.ClearMesh();
539 
540             // add light to the scene
541             DirectionalLight DirLight1 = new DirectionalLight();
542             DirLight1.Color = Colors.White;
543             DirLight1.Direction = new Vector3D(0, 0, -1);
544 
545             // add a camera to the scene
546             PerspectiveCamera Camera1 = new PerspectiveCamera();
547 
548             // set the location of the camera
549             Camera1.Position = new Point3D(160, 120, 480);
550             Camera1.LookDirection = new Vector3D(0, 0, -1);
551             Camera1.UpDirection = new Vector3D(0, -1, 0);
552 
553             // create the mesh from depth data
554             this.BuildMesh();
555 
556             // add texture to all the points
557             tmesh.Positions = corners;
558             tmesh.TriangleIndices = Triangles;
559             tmesh.Normals = Normals;
560             tmesh.TextureCoordinates = myTextureCoordinatesCollection;
561             msheet.Geometry = tmesh;
562             msheet.Material = new DiffuseMaterial((SolidColorBrush)(new BrushConverter().ConvertFrom("#↙
    52318F")));
563 
564             // build the scene and display it
565             this.modelGroup.Children.Add(msheet);
566             this.modelGroup.Children.Add(DirLight1);
567             this.modelsVisual.Content = this.modelGroup;
568             this.myViewport.IsHitTestVisible = false;
569             this.myViewport.Camera = Camera1;
570             this.myViewport.Children.Add(this.modelsVisual);
571             KinectNormalView.Children.Add(this.myViewport);
572             this.myViewport.Height = KinectNormalView.Height;
573             this.myViewport.Width = KinectNormalView.Width;
574             Canvas.SetTop(this.myViewport, 0);
575             Canvas.SetLeft(this.myViewport, 0);
576 
577         }
578 
579         /// <summary>
580         /// Export the completed mesh to a .obj file
581         /// </summary>
582         /// <param name="sender">object sending the event</param>
583         /// <param name="e">event arguments</param>
584         private void Export_Model_Click(object sender, RoutedEventArgs e)
585         {
586             //function from Helix Toolkit
587             string fileName = Model_Name.Text + ".obj";
588 
589             using (var exporter = new ObjExporter(fileName))
590             {
591                 exporter.Export(this.modelGroup);
592             }
593 
594             // test code for seeing depth frame values
595             Process.Start("explorer.exe", "/select,\"" + fileName + "\"");
596 
597             string fileName2 = "depth.txt";
598 
```

```
599             using (System.IO.StreamWriter file = new System.IO.StreamWriter(fileName2))
600             {
601                 //file.Write(string.Join(",", this.Depth));
602                 file.Write(greatestDepth);
603             }
604 
605         }
606 
607         /// <summary>
608         /// Show exception info on status bar
609         /// </summary>
610         /// <param name="message">Message to show on status bar</param>
611         private void ShowStatusMessage(string message)
612         {
613             this.Dispatcher.BeginInvoke((Action)(() =>
614             {
615                 this.ResetFps();
616                 this.KinectStatusText.Content = message;
617             }));
618         }
619 
620         /// <summary>
621         /// clear everything from the scene and canvas
622         /// </summary>
623         public void ClearMesh()
624         {
625             KinectNormalView.Children.Clear();
626             modelGroup.Children.Clear();
627             myViewport.Children.Clear();
628             modelsVisual.Children.Clear();
629             tmesh.Positions.Clear();
630             tmesh.TriangleIndices.Clear();
631             tmesh.Normals.Clear();
632             tmesh.TextureCoordinates.Clear();
633 
634         }
635 
636         /// <summary>
637         /// Clear canvas button click
638         /// </summary>
639         /// <param name="sender">object sending the event</param>
640         /// <param name="e">event arguments</param>
641         private void End_Scan_Click(object sender, RoutedEventArgs e)
642         {
643             this.ClearMesh();
644         }
645     }
646 }
```

0.2 KinectScan Graphical User Interface Code

```
<Window
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:local="clr-namespace:kinectScan"
        xmlns:sys="clr-namespace:System;assembly=mscorlib"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008" xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006" mc:Ignorable="d" x:Class="kinectScan.MainWindow"
        xmlns:tk="clr-namespace:Microsoft.Kinect.Toolkit;assembly=Microsoft.Kinect.Toolkit"
        Title="kinectScan" Height="870" Width="1028" Loaded="WindowLoaded" Closing="WindowClosing" Top="0" Left="0" Icon="Images/Kinect.ico">

    <Window.Resources>

        <ResourceDictionary Source="/KinectResources.xaml" />

    </Window.Resources>

    <Grid x:Name="LayoutGrid"  Margin="0, 0, 0, 0">

        <Grid.RowDefinitions>
            <RowDefinition />
        </Grid.RowDefinitions>

        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="700" />
            <ColumnDefinition Width="30" />
            <ColumnDefinition />
        </Grid.ColumnDefinitions>

        <Rectangle Fill="{StaticResource SecondaryBrandBrush}" />

        <Grid x:Name="CameraZone" Margin="0,0,0,0" TextBlock.FontFamily="{StaticResource KinectFont}" Grid.Column="0">

            <Grid.RowDefinitions>
                <RowDefinition Height="270" />
                <RowDefinition Height="30"/>
                <RowDefinition Height="30"/>
                <RowDefinition Height="510" />
            </Grid.RowDefinitions>

            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="700" />
            </Grid.ColumnDefinitions>

            <!-- Depth Camera -->
            <Rectangle Fill="{StaticResource MediumNeutralBrush}" Grid.Row="0" Height="240" Width="320" Margin="30,30,350,0" />
            <Image Name="KinectDepthView" Grid.Row="0" Height="240" Width="320" Margin="30,30,350, 0"/>

            <!-- Bilateral Camera-->
            <Rectangle Fill="{StaticResource MediumNeutralBrush}" Grid.Row="0" Height="240" Width="320" Margin="350,30,30,0" />
            <Image Name="KinectRGBView" Grid.Row="0" Height="240" Width="320" Margin="350,30,30,0"/>

            <!-- Reconstruction Model -->
            <Grid x:Name="Reconstruction_Grid" Grid.Row="3">
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="30"/>
                    <ColumnDefinition/>
                    <ColumnDefinition Width="30"/>
                </Grid.ColumnDefinitions>

                <Grid.RowDefinitions>
                    <RowDefinition Height="480"/>
                    <RowDefinition/>
                </Grid.RowDefinitions>

```

```
64              <Rectangle Fill="{StaticResource MediumNeutralBrush}" Grid.Row="3" Height="480" Width="640" Margin="30,0,30,30" />
65              <Canvas Name="KinectNormalView" Grid.Column="1" Height="480" Width="640" Margin="0,0,0,30" Background="{StaticResource MediumNeutralBrush}"/>
66
67              <!-- Bounding Box-->
68              <!-- <Border BorderBrush="Red" BorderThickness="1" Grid.Column="1" /> -->
69
70          </Grid>
71
72          <!-- Titles -->
73          <Label x:Name="IR_Title" Content="IR DEPTH CAMERA"  Grid.Row="1" Foreground="White"  HorizontalAlignment="Left" Margin="30,0,0,0" VerticalAlignment="Top"/>
74          <Label x:Name="RGB_Title" Content="RGB CAMERA"  Grid.Row="1" Foreground="White"  HorizontalAlignment="Right" Margin="0,0,30,0" VerticalAlignment="Top"/>
75          <Label x:Name="Model_Title" Content="RECONSTRUCTED MODEL" Grid.Row="2" Foreground="White"  HorizontalAlignment="Center" Margin="0,0,0,0" VerticalAlignment="Bottom"/>
76          <Label x:Name="statusBarText" Grid.Row="1" Foreground="White" HorizontalAlignment="Center"  Margin="0,0,0,0" VerticalAlignment="Center" Grid.RowSpan="2"/>
77          <Label x:Name="KinectStatusText" Content="Kinect Status: Loading..."  Grid.Row="3"  Foreground="White" HorizontalAlignment="Left" Margin="10,0,0,5" VerticalAlignment="Bottom"/>
78
79      </Grid>
80      <!--CameraZone-->
81
82      <Grid x:Name="MenuArea" Background="White" Grid.Column="2">
83
84          <Grid.RowDefinitions>
85              <RowDefinition Height="90" />
86              <RowDefinition Height="240" />
87              <RowDefinition Height="50" />
88              <RowDefinition />
89              <RowDefinition Height="30" />
90              <RowDefinition Height="100" />
91          </Grid.RowDefinitions>
92
93          <Grid.ColumnDefinitions>
94              <ColumnDefinition Width="290" />
95          </Grid.ColumnDefinitions>
96
97          <Button x:Name ="Begin_Scan" Content="RECORD FRAME" Margin="0,30,0,0" Style="{StaticResource KinectButton}" Grid.Row="0" Click="Begin_Scan_Click"/>
98          <Button x:Name ="End_Scan" Content="CLEAR CANVAS" Margin="137,30,0,0" Style="{StaticResource KinectButton}" Grid.Row="0" Click="End_Scan_Click" />
99
100         <!--BeginSlider Area-->
101         <Grid x:Name="SliderArea" Background="White" Grid.Row="1" Margin="0,0,30,30" Grid.RowSpan="2">
102             <Grid.RowDefinitions>
103                 <RowDefinition Height="40" />
104                 <RowDefinition Height="40" />
105                 <RowDefinition Height="40" />
106                 <RowDefinition Height="40" />
107                 <RowDefinition Height="40" />
108                 <RowDefinition Height="40" />
109                 <RowDefinition Height="40" />
110             </Grid.RowDefinitions>
111
112             <Grid.ColumnDefinitions>
113                 <ColumnDefinition Width="220" />
114                 <ColumnDefinition Width="40" />
115             </Grid.ColumnDefinitions>
116
117             <Label x:Name="Near_Filter_Title" Content="MIN FILTER DEPTH" Foreground="{StaticResource SecondaryBrandBrush}" HorizontalAlignment="Left" Margin="10,0,0,0" VerticalAlignment="Top" Grid.Row="0" Grid.Column="0" />
118             <Slider x:Name="Near_Filter_Slider" HorizontalAlignment="Left"  Margin="10,20,0,0" VerticalAlignment="Top" Width="200" Style="{StaticResource SliderStyle}" Grid.Row="0" Grid.Column=
```

```
"0" Minimum="0" Maximum="654" Value="0"/>
119             <Label x:Name="Near_Filter_Value" Content="{Binding ElementName=Near_Filter_Slider,Path
=Value}" ContentStringFormat="{}{0:N0}" Grid.Row="0" Grid.Column="1" Foreground="{StaticResource
SecondaryBrandBrush}" HorizontalAlignment="Left" VerticalAlignment="Center" />
120
121             <Label x:Name="Far_Filter_Title" Content="MAX FILTER DEPTH" Foreground="{StaticResource
 SecondaryBrandBrush}" HorizontalAlignment="Left" Margin="10,0,0,0" VerticalAlignment="Top" Grid.
Row="1" Grid.Column="0" />
122             <Slider x:Name="Far_Filter_Slider" HorizontalAlignment="Left"  Margin="10,20,0,0"
VerticalAlignment="Top" Width="200" Style="{StaticResource SliderStyle}" Grid.Row="1" Grid.Column=
"0" Minimum="{Binding ElementName=Near_Filter_Slider,Path=Value}" Maximum="654" Value="300"/>
123             <Label x:Name="Far_Filter_Value" Content="{Binding ElementName=Far_Filter_Slider,Path=
Value}" ContentStringFormat="{}{0:N0}" Grid.Row="1" Grid.Column="1" Foreground="{StaticResource
SecondaryBrandBrush}" HorizontalAlignment="Left" VerticalAlignment="Center" />
124
125             <Label x:Name="Left_Title" Content="LEFT BOUND" Foreground="{StaticResource
SecondaryBrandBrush}" HorizontalAlignment="Left" Margin="10,0,0,0" VerticalAlignment="Top" Grid.Row
="2" Grid.Column="0" />
126             <Slider x:Name="Left_Slider" HorizontalAlignment="Left"  Margin="10,20,0,0"
VerticalAlignment="Top" Width="200" Style="{StaticResource SliderStyle}" Grid.Row="2" Grid.Column=
"0" Minimum="0" Maximum="320" Value="0"/>
127             <Label x:Name="Left_Value" Content="{Binding ElementName=Left_Slider,Path=Value}"
ContentStringFormat="{}{0:N0}" Grid.Row="2" Grid.Column="1" Foreground="{StaticResource
SecondaryBrandBrush}" HorizontalAlignment="Left" VerticalAlignment="Center" />
128
129             <Label x:Name="Right_Title" Content="RIGHT BOUND" Foreground="{StaticResource
SecondaryBrandBrush}" HorizontalAlignment="Left" Margin="10,0,0,0" VerticalAlignment="Top" Grid.Row
="3" Grid.Column="0" />
130             <Slider x:Name="Right_Slider" HorizontalAlignment="Left"  Margin="10,20,0,0"
VerticalAlignment="Top" Width="200" Style="{StaticResource SliderStyle}" Grid.Row="3" Grid.Column=
"0" Minimum="0" Maximum="320" Value="320"/>
131             <Label x:Name="Right_Value" Content="{Binding ElementName=Right_Slider,Path=Value}"
ContentStringFormat="{}{0:N0}" Grid.Row="3" Grid.Column="1" Foreground="{StaticResource
SecondaryBrandBrush}" HorizontalAlignment="Left" VerticalAlignment="Center" />
132
133             <Label x:Name="Top_Title" Content="TOP BOUND" Foreground="{StaticResource
SecondaryBrandBrush}" HorizontalAlignment="Left" Margin="10,0,0,0" VerticalAlignment="Top" Grid.Row
="4" Grid.Column="0" />
134             <Slider x:Name="Top_Slider" HorizontalAlignment="Left"  Margin="10,20,0,0"
VerticalAlignment="Top" Width="200" Style="{StaticResource SliderStyle}" Grid.Row="4" Grid.Column=
"0" Minimum="0" Maximum="240" Value="0"/>
135             <Label x:Name="Top_Value" Content="{Binding ElementName=Top_Slider,Path=Value}"
ContentStringFormat="{}{0:N0}" Grid.Row="4" Grid.Column="1" Foreground="{StaticResource
SecondaryBrandBrush}" HorizontalAlignment="Left" VerticalAlignment="Center" />
136
137             <Label x:Name="Bot_Title" Content="BOTTOM BOUND" Foreground="{StaticResource
SecondaryBrandBrush}" HorizontalAlignment="Left" Margin="10,0,0,0" VerticalAlignment="Top" Grid.Row
="5" Grid.Column="0" />
138             <Slider x:Name="Bot_Slider" HorizontalAlignment="Left"  Margin="10,20,0,0"
VerticalAlignment="Top" Width="200" Style="{StaticResource SliderStyle}" Grid.Row="5" Grid.Column=
"0" Minimum="0" Maximum="240" Value="240"/>
139             <Label x:Name="Bot_Value" Content="{Binding ElementName=Bot_Slider,Path=Value}"
ContentStringFormat="{}{0:N0}" Grid.Row="5" Grid.Column="1" Foreground="{StaticResource
SecondaryBrandBrush}" HorizontalAlignment="Left" VerticalAlignment="Center" />
140         </Grid>
141         <!--EndSliderArea-->
142
143         <!--Begin Radio-->
144         <Label x:Name="Filter_Type_Title" Content="FILTER TYPE" Foreground="{StaticResource
SecondaryBrandBrush}" HorizontalAlignment="Left" Margin="10,0,0,0" VerticalAlignment="Top" Grid.Row
="2" />
145         <RadioButton Name="Filter_Off" Content="Off" HorizontalAlignment="Left" Margin="10,30,0,0"
Grid.Row="2" VerticalAlignment="Top" IsChecked="True" />
146         <RadioButton Name="Filter_Blur" Content="Blur" HorizontalAlignment="Left" Margin="60,30,0,0
" Grid.Row="2" VerticalAlignment="Top" />
147         <!--End Radio-->
148
149         <!--ModelNameArea-->
```

```
150             <Grid x:Name="ModelNameArea" Grid.Row="4">
151                 <Grid.RowDefinitions>
152                     <RowDefinition />
153                 </Grid.RowDefinitions>
154 
155                 <Grid.ColumnDefinitions>
156                     <ColumnDefinition Width="100" />
157                     <ColumnDefinition />
158                 </Grid.ColumnDefinitions>
159                 <Label x:Name="Name_Label" Content="MODEL NAME:" Foreground="{StaticResource SecondaryBrandBrush}" HorizontalAlignment="Left" VerticalAlignment="Top" Grid.Column="0" Margin="7,0,0,0" />
160                 <TextBox x:Name="Model_Name" Text="modelName"  HorizontalAlignment="Left" VerticalAlignment="Top" Width="140"  Margin="17,0,0,0" Grid.Column="1" />
161             </Grid>
162             <!--EndModelNameArea-->
163 
164             <Button x:Name ="Export_Model"  VerticalAlignment="Bottom" Margin="50,0,0,23" Style="{StaticResource KinectButton}" Grid.Row="5" Click="Export_Model_Click">
165                 <StackPanel Orientation="Horizontal">
166                     <Label x:Name="Export_Label" Content="EXPORT MODEL" Foreground="White" FontFamily="{StaticResource KinectFont}" FontSize="14" Padding="0,0,10,0"/>
167                     <Image x:Name="Download" Source="Images/download.png"  Width="23" Height="23" HorizontalAlignment="Left" VerticalAlignment="Top" />
168                 </StackPanel>
169             </Button>
170             <!-- <TextBox Name="test_text" HorizontalAlignment="Left" Height="109" Margin="42,124,0,0" Grid.Row="3" TextWrapping="Wrap" Text="TextBox" VerticalAlignment="Top" Width="209"/> -->
171         </Grid>
172         <!--MenuArea-->
173         <tk:KinectSensorChooserUI Name="sensorChooserUI" HorizontalAlignment="Center" Margin="330,0,330,5"/>
174     </Grid>
175     <!--LayoutGrid-->
176 
177 </Window>
```

References

[1] How kinect and kinect fusion (kinfu) works, December 2011.

[2] Helmut Grabner Xiaofeng Ren-Kurt Konolige Andrea Fossati, Juergen Gall, editor. *Consumer Depth Cameras for Computer Vision Research Topics and Applications*. Springer-Verlag London, 2013.

[3] Richard A. D'Aveni. 3-d printing will change the world. *Harvard Business Review*, 2013.

[4] Lesliei Gordon. The changing face of 3d printing. *Machine Design*, 2013.

[5] Satoshi Tanaka Soo-Hyun Park Jong-Hyun Kim, Kangsun Lee. *Advanced Methods, Techniques, and Applications in Modeling and Simulation*. Springer Japan, 2012.

[6] Ashlee Vance. 3-d printing spurs a manufacturing revolution. *New York TImes*, 2010.