ESCÁNER 3D PARA RECONSTRUCCIÓN COMPUTARIZADA DE OBJETOS REALES

Romero Hernández Antonio de Jesús[1], García de la Rosa Luis Armando[2]

[1]Ingeniería en Mecatrónica, Instituto Tecnológico Superior de Guanajuato | chuy_18vl@hotmail.com

Ingeniería en Sistemas Computacionales, Instituto Tecnológico Superior de Guanajuato | lgarcia@itesg.edu.mx

Resumen

Se programó una aplicación en C# que captura y procesa el buffer de profundidad del sensor Kinect de Microsoft. Dicha aplicación despliega la reconstrucción por computadora de cualquier objeto frente del Kinect, por medio del mallado de la nube de puntos capturada sobre la superficie del objeto a digitalizar. El procesamiento del mallado es muy denso, por lo que se requiere el auxilio de una tarjeta aceleradora de gráficos. Adicionalmente, se diseñó y construyó una base giratoria que permite girar los objetos y capturarlos en diferentes caras y así tener la información del volumen completo. Con este prototipo se cuenta con un escáner de bajo costo y un desempeño mediano a alto que ayuda al modelado de piezas mecánicas. Sin embargo, se requiere seguir trabajando en el mallado, ya que dependiendo del objeto, la técnica empleada obtiene escasos resultados.

Abstract

An application was programmed in C #, that captures and process the depth buffer of a Microsoft Kinect sensor. This application displays the computer reconstruction of any object in front of the Kinect, through the meshing of the 3D point cloud captured on the surface of the object to be scanned. Processing the mesh is very dense, so to help, a graphics accelerator card is required. Additionally, we design and built a rotationary base that allows you to rotate objects and capture the different faces that constitute the object in order to have the full volume information. With this prototype, we have a scanner of low cost and medium-high performance that helps the modeling of mechanical parts. However, more work is needed on the meshing, because depending on the object, the technique used, gets poor results.

Palabras Clave Kinect, Profundidad, Computadora, Procesamiento, Diseño.

INTRODUCCIÓN

Las tecnologías de escaneo están basadas en el barrido 2D. Estos aparatos tienen fuentes de luz, las cuales sirven para capturar lo datos de una imagen en 2D. Estos equipos tienen un emisor y un receptor, el trabajo de estos aparatos es que lanzan disparos en una cierta área de trabajo, primero rastrean de forma horizontal línea por línea, cuando termina, continua de manera vertical de la misma forma, ya que estos dispositivos detectan pixel por pixel (un pixel es la mínima unidad que se puede detectar en una imagen). Por otra parte, los escáneres 3D hacen uso de una solución parecida a la que se emplea en un escáner 2D. La única diferencia, consiste en que el sensor es lo suficientemente sensible para capturar información distinta a la intensidad de la luz empleada en el barrido. Esta información puede ser: Flujo óptico, emisidad, fase, etc. El problema radica en la cantidad de información que se tiene que procesar, pues entre más chico el pixel mayor información se generará en el barrido.

Los sensores más comúnmente usados para los escáneres ópticos son: sensores de campo eléctrico, diagonal y magnético, así como, cámaras ópticas, infrarrojas y CCD. Tanto los sensores de campo y las cámaras CCD son relativamente costosas, puesto que requieren equipamiento especial para su funcionamiento, adicionalmente, existen componentes y módulos de software ya desarrollados para poder hacer uso de las mismas. Sin embargo, las cámaras ópticas e infrarrojas son relativamente más baratas que las tecnologías anteriormente ya mencionadas. En cambio es necesario programar gran cantidad de código en orden de poder obtener información de ellas, lo que representa una desventaja frente al costo.

Entre la gran variedad de cámaras y sensores disponibles en el mercado encontramos el sensor Kinect de Microsoft. El cual permite la detección de diferentes flujos como son: Video RGB, flujo de profundidad y esqueleto, de hasta 4 personas. El video RGB es un flujo de video a color con resoluciones disponibles de 20x240px en formato de 16 y 32bpp corriendo a 30fps. El flujo de profundidad se origina por el sensor infrarrojo incluido en el Kinect y permite obtener información tridimensional de la escena que ve la cámara RGB. El flujo de esqueleto es un rastreo que realiza el sensor de 20 puntos de unión que representa articulaciones del cuerpo humano, con lo cual se puede capturar movimiento de las personas.

OBJETIVOS

General:

Generar un modelo tridimensional de un objeto real, por medio del Kinect, para su posterior manipulación en computadora.

Específicos:

Aprender a programar en C#.

Aprender a usar las librerías de desarrollo de Kinect para Windows.

Caracterización del Kinect.

Diseñar y desarrollar Base Giratoria.

MATERIALES Y MÉTODOS

Equipamiento

Para la realización del proyecto se empleó el equipamiento que se muestran en la tabla 1. Es importante destacar que el sensor Kinect empleado es la versión del Xbox 360, ya que fue imposible conseguir la versión para Windows. La tarjeta gráfica tiene la característica de tener 2Gb en RAM y 1024 GPUs.

Tabla 1 Materiales empleados en el proyecto.

Materiales
1. Tarjeta gráfica NVIDIA GTX 750T1
2. Computadora
3. Arduino UNO
4. Sensor Kinect de Microsoft más librerías de desarrollo (SDK)
5. IDE para C#

6. Motor a pasos

Metodología

Primeramente se realizó una búsqueda bibliográfica sobre el funcionamiento del Kinect. Con lo cual se pudo saber y entender cómo funciona el Kinect y los requisitos para el desarrollo de aplicaciones Windows. En seguida se procedió a familiarizarse con el entorno de desarrollo integrado (IDE) para C# por medio de la codificación de programas sencillos, en nuestro caso escogimos el IDE, SharpDevelop, ya que es de código abierto y de bajo consumo de recursos. Posteriormente se integró el Kit de Desarrollo de software (SDK) de Kinect para Windows, programando los ejemplos que distribuye Microsoft en el libro de Guía (Abhijit Jana, Kinect for Windows SDK Programming Guide). Posteriormente, ya con la experiencia adquirida se diseñó y se programó una aplicación que captura el flujo de profundidad (depth stream), sobre el cual se generó una nube de puntos 3D que pertenecen a la superficie del objeto escaneado; una vez obtenida la nube de puntos se renderizó una superficie interpolada, la cual corresponde aproximadamente al objeto escaneado. Haciendo este procedimiento con las distintas caras del objeto, es posible reconstruir el volumen por completo. Es importante señalar que una gran parte de procesamiento de renderizado se tiene que llevar a cabo en una tarjeta aceleradora de gráficos ya que es muy denso y hace que la computadora se vuelva muy lenta o inclusive se bloqueé. Finalmente se diseñó la una base giratoria para auxiliar al programa en la captura de las distintas caras de los objetos. Adicionalmente, se automatizó el movimiento de la base giratoria por medio de arduino para que así con un solo click se pueda realizar la digitalización completa de un objeto.

RESULTADOS Y DISCUSIÓN

Como resultado podemos señalar que se programó una interfaz Windows que por medio de un botón realiza la digitalización de objetos reales. En la Fig. 1a podemos apreciar parte del código C# de la aplicación. Mientras que en la Fig. 1b podemos ver la interfaz Windows.

```
33
34      private void button1_Click(object sender, RoutedEventArgs e)
35      {
36          DirectionalLight DirLight1 = new DirectionalLight();
37          DirLight1.Color = Colors.White;
38          DirLight1.Direction = new Vector3D(1, 1, 1);
39
40
41          PerspectiveCamera Camera1 = new PerspectiveCamera();
42          Camera1.FarPlaneDistance = 8000;
43          Camera1.NearPlaneDistance = 100;
44          Camera1.FieldOfView = 10;
45          Camera1.Position = new Point3D(160, 120, -1000);
46          Camera1.LookDirection = new Vector3D(0, 0, 1);
47          Camera1.UpDirection = new Vector3D(0, -1, 0);
48
49          Model3DGroup modelGroup = new Model3DGroup();
50
51          int i = 0;
52          for (int y = 0; y < 240; y += s)
53          {
54              for (int x = 0; x < 320; x += s)
55              {
56                  points[i] = Triangle(x, y, s);
57                  // points[i]=MCube(x,y);
58                  points[i].Transform = new TranslateTransform3D(0, 0, 0);
59                  modelGroup.Children.Add(points[i]);
60                  i++;
61              }
62          }
63
64          modelGroup.Children.Add(DirLight1);
65          ModelVisual3D modelsVisual = new ModelVisual3D();
66          modelsVisual.Content = modelGroup;
67          Viewport3D myViewport = new Viewport3D();
68          myViewport.IsHitTestVisible = false;
```

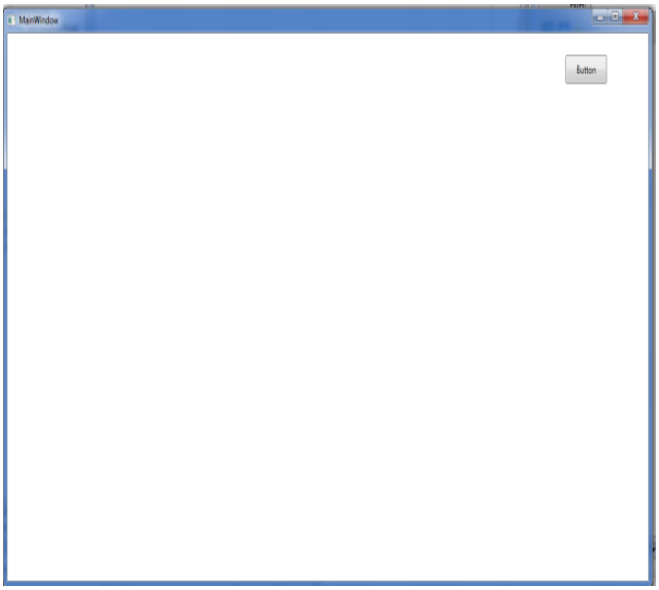

Fig.1. a) Parte del código. b) Interfaz Windows de la aplicación desarrollada.

Por otra parte en la Fig. 2, se muestra una sección de un objeto renderizado por medio de nuestra aplicación. Es importante señalar que se empleó el mallado triangular y que el modelo 3D empleado para guardar la reconstrucción es un simple obj. El cuál solo contiene la información de vértices y de la textura.

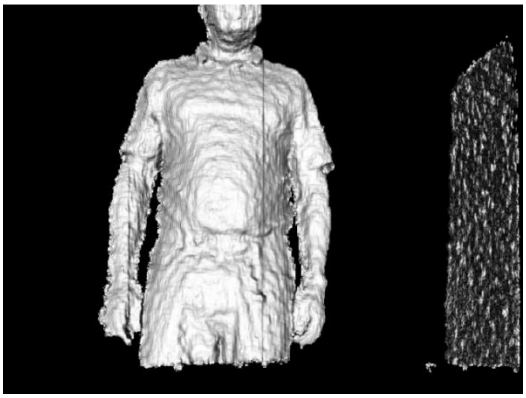

Fig.2. Renderizado de una sola cara de un objeto.

El sensor Kinect tiene algunas limitaciones como el alcance de captura, el cual en nuestras pruebas era de 30 a 180 cm, pero encontramos que es más óptimo alrededor de los 100cm. También pudimos encontrar que se necesita un lugar de trabajo con poca luz, ya que de lo contrario el sensor infrarrojo registra mucho ruido. Por otra parte el consumo de recursos computacionales en el renderizado es excesivo ya que se manejan una gran cantidad de voxels o puntos 3D (alrededor de 100,000). Finalmente, la técnica de renderizado afecta profundamente al desempeño de nuestra aplicación ya que muestra más o menos detalles del objeto, dependiendo de la cantidad de caras del mismo, por lo que podemos concluir que la técnica empleada, realmente está muy limitada. Se pretende continuar trabajando con la incorporación de otras técnicas (splines, Bezier, etc.), para aumentar el desempeño y mejorar el prototipo.

CONCLUSIONES

Los objetivos se cumplieron completamente. Actualmente se cuenta con un escáner 3D de mediano desempeño a un bajo costo capaz de digitalizar objetos reales de pocas caras. Existe una dificultad mediana para programar en C#, sin embargo, tener experiencia en programar ayuda a superar estas dificultades, el SDK del Kinect cuenta con una gran cantidad de funciones que permiten hacer uso de todas las prestaciones del Kinect y sacarle más jugo como sensor, sin embargo, el problema aquí radica en saber aprovechar esas funciones para nuestro beneficio. La técnica de mallado afecta al desempeño de la superficie renderizada, por lo que es conveniente

emplear distintas y verificar cual es la más apropiada para nuestro objeto. Además la renderización es muy costosa computacionalmente hablando, por lo que es necesario un sistema de cómputo robusto, lo que lleva a pensar en emplear computo de alto desempeño para poder asumir dicho costo y lograr un producto de mayor calidad.

AGRADECIMIENTOS

Se agradece al CONCYTEG por el financiamiento para el proyecto y al Dr. Luis Armando García de la Rosa por darme la oportunidad de colaborar en dicha investigación.

REFERENCIAS

Gonzales Seco, José Antonio, "El lenguaje de programación C#", Introducción al C#, pg. 21 a 30. "El lenguaje de programación C#", Interfaces, pg. 171 a 176. "El lenguaje de programación C#", Instrucciones, pg. 179 a 204.

Abhijit Jana, "Kinect for Windows SDK Programming Guide", Getting the depth and player index automatically a 3D view of depth data, pg.144 a 155.

Practical –windows-kinect-in-c, http://www.i-programmer.info/ebooks/4126-kinect-sdk1-a-3d-point-cloud.html

3D-Point-Cloud-With-The-Kinect, http://chanel19.msdn.com/coding4fun/kinect