División de Ingenierías Campus Irapuato-Salamanca.

Manual para la reconstrucción de escenarios 3D mediante nube de puntos en C++ con Kinect 360 y OpenGL (freeGLUT).

Elaborado por:

Eduardo de Jesús Gasca Laguna. Lic. En Ing. en Comunicaciones y Electrónica.

Jonathan Duarte Jasso Lic. en Ing. en Mecatrónica.

Juan Pablo Gonzalez Mendoza Lic. en Ing. en Mecatrónica.

Francisco Javier Serrano Martínez Lic. en Ing. en Mecatrónica.

Asesora: Dra. Dora Luz Almanza Ojeda.

Índice

¿Qué es un dispositivo RGBD?

De manera muy simple y general. Un dispositivo o sensor RGBD consiste en una cámara RGB y un sensor de profundidad basado en infrarrojos. La cámara RGB no es más que una cámara común como las cámaras web y el sensor de profundidad consta de un emisor de infrarrojos que hace incidir luz infrarroja sobre los objetos y un receptor que capta estas reflexiones y mediante los tiempos de emisión y recepción hace un cálculo de a que distancia se encuentran los objetos. Hay que considerar sus limitantes como en objetos fuera de rango, superficies transparentes, etc.

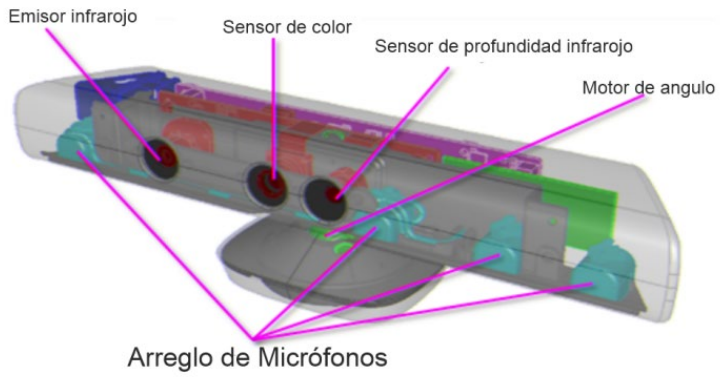


Figura 1. Sensor Kinect 360 y sus componentes.

En este manual se hace uso del dispositivo RGBD Kinect 360 lanzado por la empresa Microsoft en el año 2010 para la consola Xbox 360 (ver figura 1).

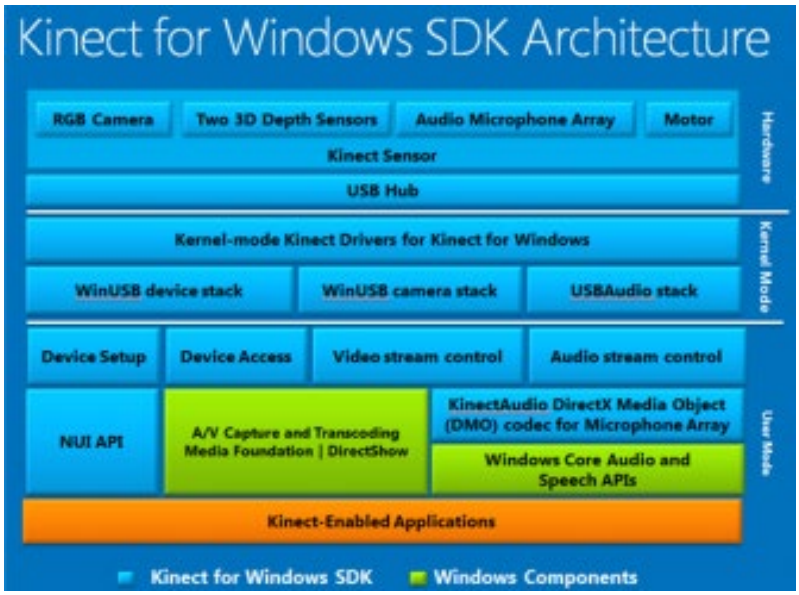
La cámara RGB (llamado a veces también como sensor de color), soporta hasta 30 cuadros por segundo a una resolución de 640x480 pixeles, el máximo de resolución es de 1280x960. El rango del ángulo de la cámara es de 43 grados en vertical y 57 en horizontal. El emisor y sensor de profundidad en matriz de puntos infrarrojos permite obtener la distancia entre un punto determinado de un objeto o escenario y la cámara. Soporta resoluciones de 640x480, 320x240 y 80x60 pixeles.

Kinect SDK para Windows

La arquitectura del SDK (Software Development Kit) para Kinect, se conforma de tres partes principales:

- Hardware: Funcionamiento en el control del hardware, sensores, cámara, el arreglo de micrófonos, conexión USB y los motores que permiten el enfoque.

- Kernel Mode: En este nivel se supervisa la comunicación y carga de los datos sin ningún tipo de procesamiento que los sensores detectan.
- User Mode: Proporciona la interacción entre aplicación y usuario final, enmarca y oculta los procesos para el análisis de los datos en crudo, otorga al programador una interfaz adecuada para el tratamiento de datos.



*En este proyecto se hace uso de “Kinect for Windows SDK v1.8” sobre Windows 10

Para más información puede visitar el sitio [1].

OpenGL

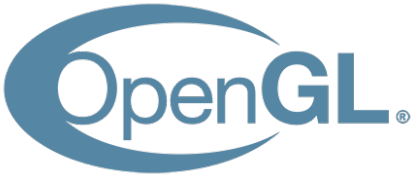


Figura 2. Logotipo de la librería OpenGL.

OpenGL, es una especificación estándar que define una API (application programming interface) para la generación de gráficos en 2D y 3D. Consiste en 250 diferentes funciones que pueden usarse para dibujar escenas tridimensionales complejas a partir de primitivas geométricas simples, tales como puntos, líneas y triángulos, fue desarrollada originalmente por Silicon Graphics Inc. en 1992.

Decir que es una especificación, se refiere a que es un documento que describe un conjunto de funciones y el comportamiento exacto que deben tener. Partiendo de ella, los fabricantes de hardware crean implementaciones, que son bibliotecas de funciones que se ajustan a los requisitos de la especificación, utilizando aceleración hardware cuando es posible.

Bibliotecas y utilidades

Se han programado varias bibliotecas externas que añaden características no disponibles en el propio OpenGL.

GLU: Ofrece funciones de dibujo de alto nivel basadas en primitivas de OpenGL. Las funciones de GLU se reconocen fácilmente pues todas empiezan con el prefijo glu.

GLUT: API multiplataforma que facilita una rudimentaria funcionalidad para el manejo de ventanas e interacción por medio de teclado y ratón.

GLUI: Interfaz de usuario basada en GLUT; proporciona elementos de control tales como botones, cajas de selección y spinners. Es independiente del sistema operativo, sustentándose en GLUT para manejar los elementos dependientes del sistema.

Las influencias principales de OpenGL en la construcción de hardware son

- Primitiva básica de puntos, líneas y polígonos rasterizados.
- Pipeline de transformación e iluminación.
- Z-buffering
- Mapeo de texturas
- Alpha blending

FreeGLUT

Es una alternativa de código abierto a la biblioteca OpenGL Utility Toolkit (GLUT). GLUT (y por lo tanto FreeGLUT) permite al usuario crear y administrar ventanas que contienen contextos OpenGL.

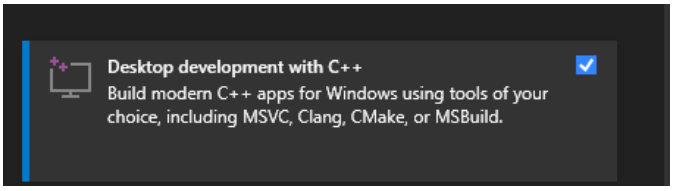
Figura 3. Logotipo de la librería freeglut.

Para más información puede visitar el sitio de la documentación oficial de OpenGL [2].

Requisitos y configuración del proyecto en Visual Studio.

Descargas e instalaciones requeridas.

1. IDE: Visual Studio Community Edition. En el instalador se debe seleccionar la casilla de desarrollo con C++ [3].

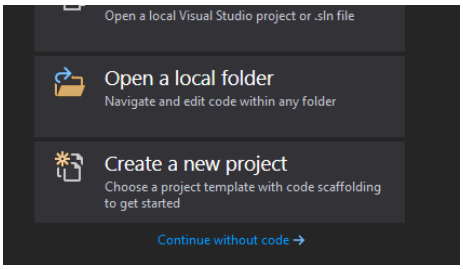


2. Librería de desarrollo para Kinect [4].
3. Librería freeGLUT para Windows [5].

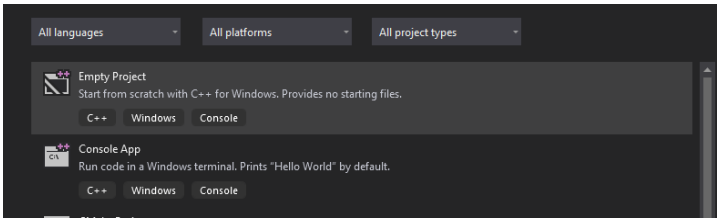
Para los dos primeros puntos no tiene mucho misterio solo ejecute los instaladores y siga las instrucciones de los instaladores. En el caso de freeglut al estar en la página de descarga, ponga atención en descargar el de la sección llamada "**freeglut 3.0.0 MSVC Package**", se descarga un archivo zip. Extraiga el contenido y recuerde bien la ruta de donde coloca esta carpeta ya que será de suma importancia para su configuración. Se sugiere deje el contenido directamente en el disco C. De manera que quede la ruta de ubicación de la siguiente manera C:\freeglut

Configuración del IDE.

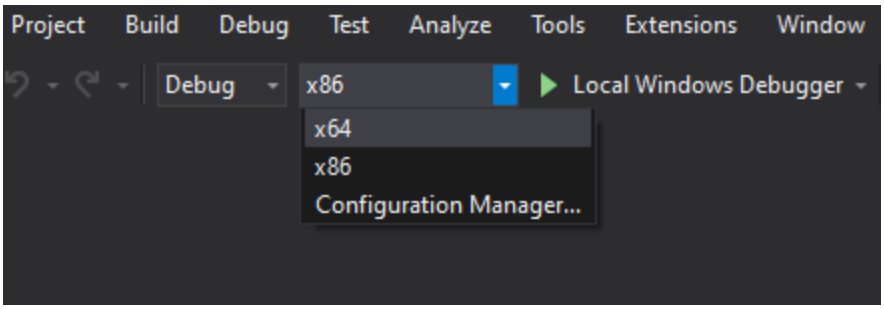
1. Abrimos Visual Studio y seleccionamos en crear nuevo proyecto.



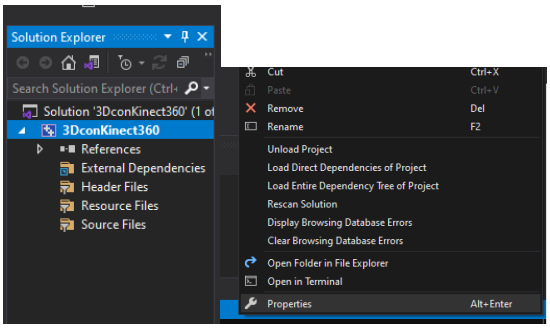
2. Seleccionamos crear proyecto vacío.



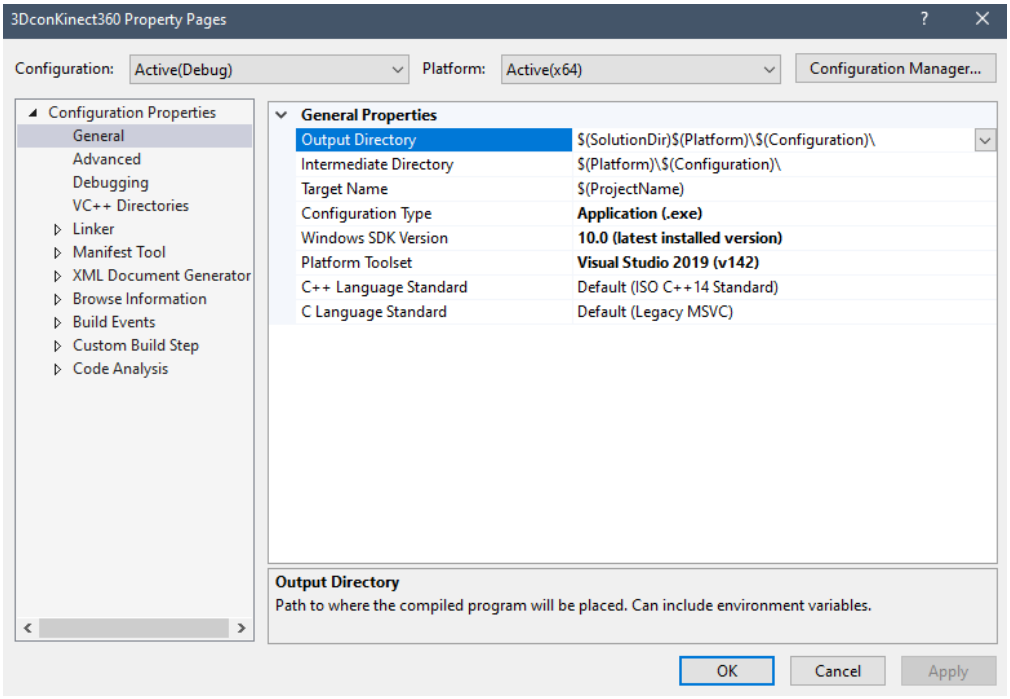
3. Nombramos nuestro proyecto y elegimos la ruta donde se creará nuestra solución de proyecto. Y damos clic en crear.
4. En la parte superior cambiamos el Debug de x86 a x64.



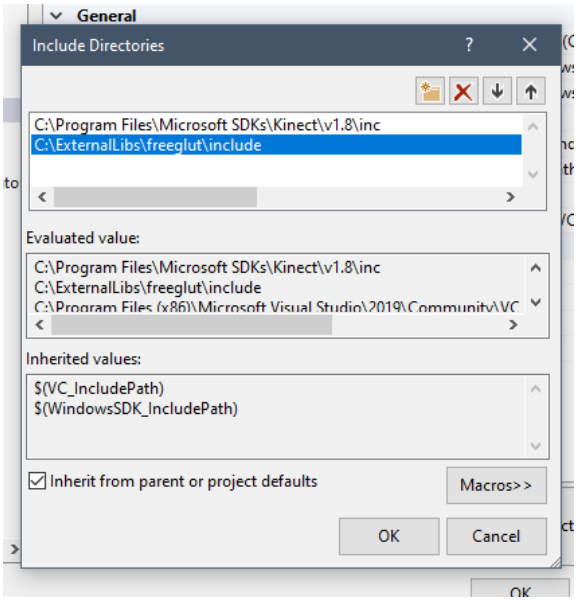
5. Damos clic derecho en el nombre de nuestro proyecto en el explorador de solución y damos clic en propiedades.



6. Dentro de la ventana de propiedades vamos a realizar las siguientes modificaciones.

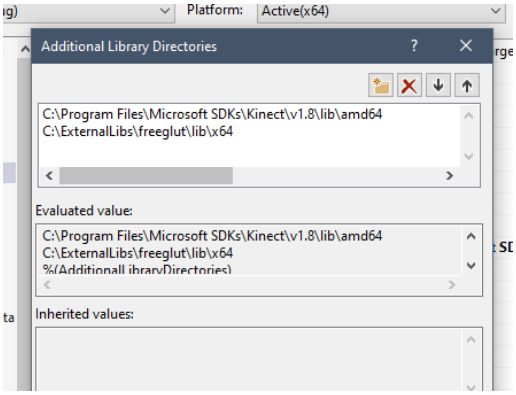


- VC++ Directories >> Include Directories >> ; Aquí agregamos las carpetas
  - C:\Program Files\Microsoft SDKs\Kinect\v1.8\inc
  - ($RUTA_DEL_USUARIO)\freeglut\include



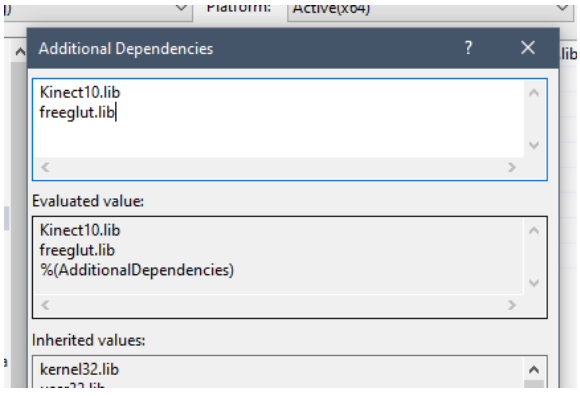
- Linker >> General>> Additional Library Directories; Agregamos
  - C:\Program Files\Microsoft SDKs\Kinect\v1.8\lib\amd64
  - ($RUTA_DEL_USUARIO)\freeglut\lib\x64

    *Recuerde tener en consideración la arquitectura del procesador.



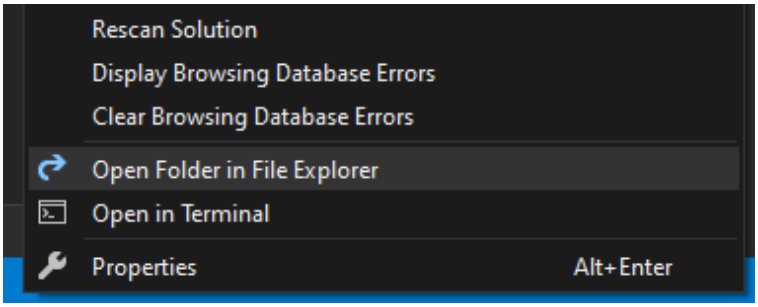
- Linker >> Input>> Additional Dependencies; Escribimos

  Kinect10.lib
  freeglut.lib

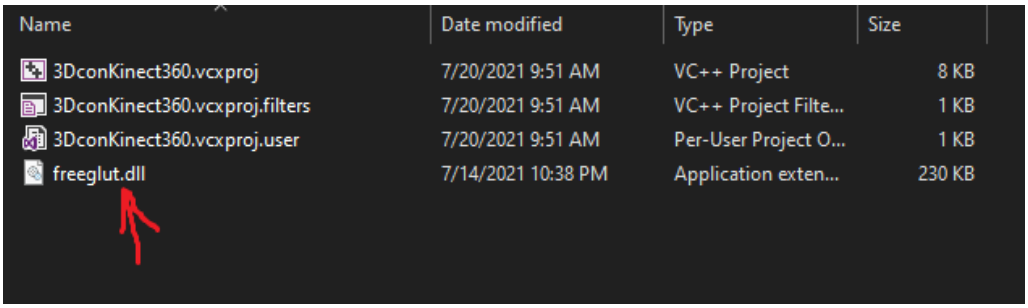


Aplicamos y Ok.

Como ultimo paso vamos a agregar la librería dinámica de freeglut.dll la cual se encuentra en la carpeta **($RUTA_DEL_USUARIO)\freeglut\bin\x64** la vamos a copiar dentro de la misma ruta donde se encontraran los archivos fuente de nuestro programa. Para ello damos clic derecho en el nombre de nuestro proyecto en el explorador y seleccionamos “Abrir carpeta en el explorador”

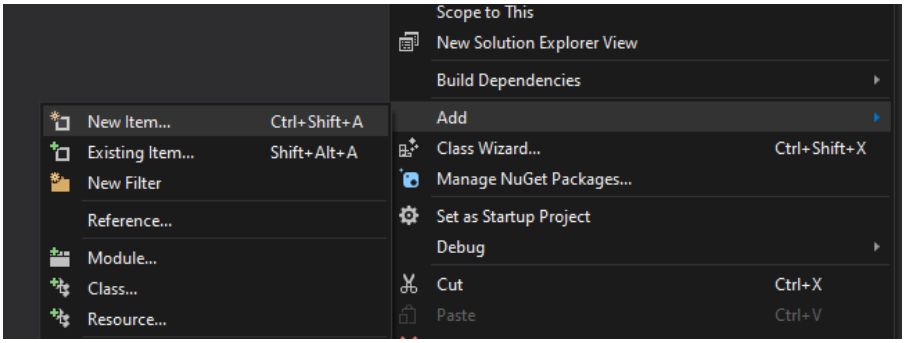


Y pegamos freeglut.dll en dicha dirección.

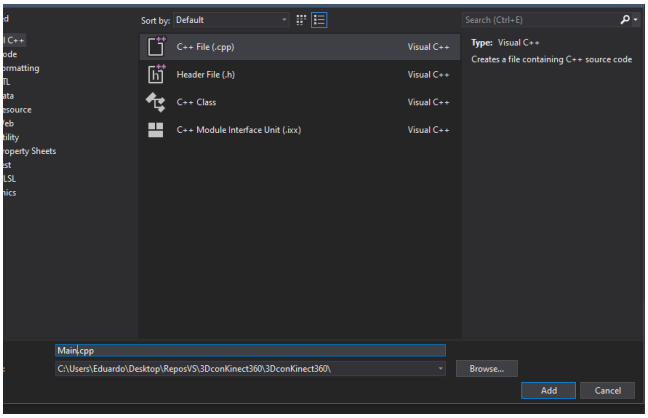


Listo ya tenemos configurado nuestro IDE para hacer uso de las librerías y compile correctamente. Procedemos a crear nuestro código en C++.

Creamos nuestro archivo .cpp dando clic derecho sobre el nombre del proyecto add>> New item



Seleccionamos C++ File, lo nombramos con al extensión .cpp y lo agregamos.



Código

En este apartado daremos una breve explicación sobre la estructura general del código y su funcionamiento.

La primera parte del código consta de las llamadas a los archivos de cabecera de las librerías y la declaración de variables globales que vamos a usar en nuestro programa.

```
//Librerias para OpenGL
#include <Windows.h>
#include <Ole2.h>
#include <gl/GL.h>
#include <gl/GLU.h>
#include <gl/glut.h>

//Libreria kinect
#include <NuiApi.h>

//Estándar C++
#include <iostream>
using namespace std;

//Resolucion
#define ancho 640
#define alto 480

//Variables OpenGL
GLuint texturaID;
GLubyte datos[ancho * alto * 4];

long mapeo_ProfaRGB[ancho * alto * 2]; //Matriz(Arreglo) para el solapamiento
float RGBArray[ancho * alto * 3]; //Matriz(Arreglo) de datos de color RGB
float ProfArray[ancho * alto * 3]; //Matriz(Arreglo) de datos de profundidad

//Kinect
HANDLE camaraRGB;
HANDLE sensorDeProfundidad;
```

```
INuiSensor* Kinect360;
```

Se crean las siguientes funciones.

```
bool iniciarOpenGL(int argc, char* argv[])
bool iniciarKinect()
void obtenerProfundidad(GLubyte* matriz)
void obtenerRGB(GLubyte* matriz)
void obtenerDatosKinect()
void rotar()
void mostrarDatosKinect()
void dibujar()
void lanzar()
```

*Se omite la función main ya que esta siempre debe existir en un programa C++

Funciones:

Por practicidad aquí se menciona que hace cada función asi como algunos datos importantes o detalles a tomar en cuenta. Puede ver el código completo o descargar desde GITHUB aquí

Inicialización de instancias para OpenGL. (`bool iniciarOpenGL(int argc, char* argv[])`**)**

Se declara una función de tipo booleano para inicializar los datos que serán cargados mediante OpenGL, en el DisplayMode se declara que habrá datos de profundidad, dobles y de RGBA. Se define el tamaño de ventana y se crea un nombre para la ventana. Se llama a la función dibujar que es la que desplegara los datos. Si la función regresa un FALSE quiere decir que algo no cargo correctamente. En esta parte del proceso.

Inicializacion del Kinect(`bool iniciarKinect()`**)**

Se hace el llamado al sensor Kinect contando de cuantos activos se disponen asi como a su indexación en la memoria. Se inicializa mediante *NuiInitialize*. Vamos a usar los sensores de profundidad y la cámara RGB por lo cual son los únicos que se inicializan con el método *NuiImageStreamOpen* verifique que los datos para iniciar sean los correctos o los que usted requiere para trabajar.

Datos de profundidad(`void obtenerProfundidad(GLubyte* matriz)`**).**

Esta función obtiene los datos desde el sensor de profundidad al cual nosotros en las variables globales llamamos *sensorDeProfundidad* en dicha función se realiza el proceso de coordenadas homogéneas para la correcta colocación de los pixeles mediante la función Vector4. Y con la función *NuiImageGetColorPixelCoordinatesFromDepthPixelAtResolution* se guardan en profARGB que es la matriz donde colocara el color en la geometría correcta mezclando las matrices de datos de los sensores.

Datos de color RGB(`void obtenerRGB(GLubyte* matriz)`**).**

Obtiene una imagen a color mediante la cámara RGB y almacena en la matriz profARGB el color en la coordenada correcta para después mezclar los datos con los de profundidad. Y asi desplegar el color en el punto que corresponde.

Datos del Kinect(`void obtenerDatosKinect()`**).**

Realiza el mapeado del escenario, para ello llama a las funciones que obtienen la profundidad y los datos RGB para almacenarlos en el correspondiente arreglo.

MostrarDatosKinect(`void mostrarDatosKinect()`**).**

Manda la información de los datos obtenidos desde el Kinect a los Buffers de OpenGL para su visualización. Para ello usamos Buffer de color (GL_COLOR_BUFFER_BIT) y el buffer de profundidad conocido también como z-Buffer (GL_DEPTH_BUFFER_BIT). Como la visualización es atravez de una nube de puntos se declara en el glBegin(GL_POINTS) .

Rotación del modelo 3D(`void rotar()`**).**

Simplemente rota el escenario creado para poder apreciar todas las perspectivas asi como ver mas apropiadamente los detalles que se capturaron.

Función "dibujar"(`void dibujar()`**).**

Llama a la función de mostrar datos del Kinect para desplegarlos, la función glutSwapBuffers() intercambia los buffers en la ventana cuando se tiene doble que es nuestro caso.

Función "lanzar"(`void lanzar()`**).**

Únicamente contiene a la función glutMainLoop(). Esta función ingresa al ciclo de procesamiento de GLUT. Esta rutina debe llamarse solo una vez.

Main(`int main(int argc, char* argv[])`**).**

Todo programa escrito en C++ debe contener una función main, ya que esto le indica al compilador por donde comenzar a ejecutar el programa. Los dos primeros if, sirven como protección para detener el programa en caso de que OpenGL no inicie correctamente o que no se detecte o pueda iniciar un dispositivo Kinect.

Una vez realizada esta comprobación, se inicializan los valores de los buffers tanto el de color(glClearColor), como el de profundidad(glClearDepth).

gleViewport() establece los valores de la ventana, las coordenadas relativas dentro de la ventana, asi como su tamaño y posición.

glMatrixMode() el tipo de matriz que se va a desplegar.

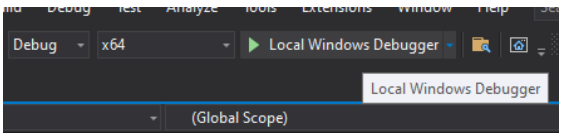
glLoadIdenty() remplaza la matriz actual con una matriz identidad.

gluPerspective() configura la matriz en perspectiva (grados en y, relación de apecto en el campo de visión, punto z mas cercano, punto z mas lejano).

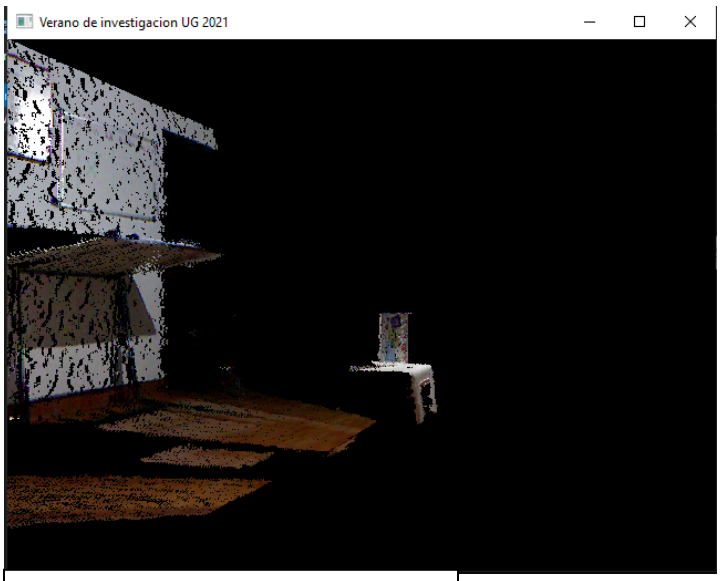
gluLookAt() Define la transformación de la visualización.

Compilado y ejecución

Para compilar y correr el programa únicamente presione F5 o de clic sobre el icono de “play” en la parte superior. Si todo es correcto el compilador no regresara errores y se ejecutara la ventana mostrando el escenario en 3D.



Captura de escenario Frontal



Captura de escenario lateral

La reconstrucción del escenario se realiza en tiempo

REFERENCIAS

[1] Documentación y API Kinect (2021). Consultado el 10 de julio de 2021 en https://docs.microsoft.com/en-us/previous-versions/windows/kinect-1.8

[2] OpenGL 2.1 Reference Pages (2021). Consultado el 1 de julio de 2021 en https://www.khronos.org/registry/OpenGL-Refpages/gl4/

[3] Visual Studio IDE (2021). Consultado el 1 de julio de 2021 en Visual Studio

[4] SDK para Kinect 360 (2021). Consultado el 6 de julio de 2021 en Kinect for Windows SDK v1.8

[5] Librería FreeGlut (2021). Consultado el 8 de julio de 2021 en FreeGlut