Sarah Wood

Embedded Programming

Chapter 8

Assignment 2

Use the TVout library to visualize some other sensor’s data. You can try

to build the parking-distance control from Chapter 5, *Sensing the World*

*Around Us*, on page 77, using the TVout library.

I used the Nintendo Nunchuk controller, since I’d had good success with it, and it was still out on the workbench. I found some older code to output a simple wireframe cube onto a TV at paulsarduino.co.uk , updated it and adapted to this exercise.

Screen Caps/ Pictures



Source Code (I did not include the Nunchuck headers, but the 3d cube stuff is both new and awesome. )

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

lib3d.h

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

#ifndef lib3d\_h

#define lib3d\_h

#define MAX\_LINES 10

class Point3D

{

public:

short x;

short y;

short z;

};

class Line3D

{

public:

Line3D();

Line3D(short x1, short y1, short z1, short x2, short y2, short z2);

Point3D p1;

Point3D p2;

};

class Scene3D

{

public:

Scene3D();

void CreateScene(short resX, short resY, TVout\* pTVout);

void UpdateDisplay();

void AddLine(Line3D line);

void AddCube(short cX, short cY, short cZ, short edgeLen);

void AddSphere(float r, Point3D c);

void SetVAngle(float vNew);

void SetHAngle(float hNew);

private:

TVout\* pTV;

Line3D lines[MAX\_LINES]; // all the lines in a 3D wireframe model

short iNumLines;

short mv; // middle vertically

short mh; // the horizontal middle

float h; // the angles

float v;

float cosh; // precalculated trig values for xyztox and xyztoy functions

float sinh;

float cosv;

float sinv;

int xyztox(short x, short y, short z);

int xyztoy(short x, short y, short z);

void UpdateConsts();

};

#endif

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

lib3d.cpp

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

#include <TVout.h>

#include "stdlib.h"

#include "math.h"

#include "lib3d.h"

Line3D::Line3D()

{

Line3D(0,0,0,0,0,0);

}

Line3D::Line3D(short x1, short y1, short z1, short x2, short y2, short z2)

{

p1.x = x1;

p1.y = y1;

p1.z = z1;

p2.x = x2;

p2.y = y2;

p2.z = z2;

}

Scene3D::Scene3D()

{

pTV = (TVout\*)0;

iNumLines = 0;

mv = 0; // middle vertically

mh = 0; // the horizontal middle

h = 0;

v = 0;

UpdateConsts();

}

void Scene3D::CreateScene(short resX, short resY, TVout\* pTVout)

{

pTV = pTVout;

iNumLines = 0;

mv = resX / 2; // middle vertically

mh = resY / 2; // the horizontal middle

h = 0;

v = 0;

UpdateConsts();

}

void Scene3D::SetVAngle(float vNew)

{

v = vNew;

UpdateConsts();

}

void Scene3D::SetHAngle(float hNew)

{

h = hNew;

UpdateConsts();

}

void Scene3D::UpdateConsts()

{

cosh = cos(h);

sinh = sin(h);

cosv = cos(v);

sinv = sin(v);

}

int Scene3D::xyztox(short x, short y, short z)

{

return mh + round(x\*cosh+z\*sinh);

}

int Scene3D::xyztoy(short x, short y, short z)

{

return mv + round(y\*cosv+(-x\*sinh+z\*cosh)\*sinv);

}

void Scene3D::AddLine(Line3D line)

{

if (iNumLines < MAX\_LINES)

{

lines[iNumLines] = line;

iNumLines++;

}

}

void Scene3D::AddCube(short cX, short cY, short cZ, short edgeLen)

{

Line3D l;

short iOffs = edgeLen - 1;

// top square first

l.p1.z = cZ + iOffs; l.p2.z = cZ + iOffs;

l.p1.x = cX - iOffs; l.p1.y = cY + iOffs;

l.p2.x = cX - iOffs; l.p2.y = cY - iOffs;

AddLine(l);

l.p1.x = cX + iOffs; l.p1.y = cY - iOffs;

AddLine(l);

l.p1.x = cX + iOffs; l.p1.y = cY + iOffs;

l.p2.x = cX + iOffs; l.p2.y = cY - iOffs;

AddLine(l);

l.p2.x = cX - iOffs; l.p2.y = cY + iOffs;

AddLine(l);

// bottom square second

l.p1.z = cZ - iOffs; l.p2.z = cZ - iOffs;

l.p1.x = cX - iOffs; l.p1.y = cY + iOffs;

l.p2.x = cX - iOffs; l.p2.y = cY - iOffs;

AddLine(l);

l.p1.x = cX + iOffs; l.p1.y = cY - iOffs;

AddLine(l);

l.p1.x = cX + iOffs; l.p1.y = cY + iOffs;

l.p2.x = cX + iOffs; l.p2.y = cY - iOffs;

AddLine(l);

l.p2.x = cX - iOffs; l.p2.y = cY + iOffs;

AddLine(l);

// 4 corner lines

l.p1.z = cZ - iOffs; l.p2.z = cZ + iOffs;

l.p1.x = cX + iOffs; l.p1.y = cY + iOffs;

l.p2.x = cX + iOffs; l.p2.y = cY + iOffs;

AddLine(l);

l.p1.x = cX - iOffs;

l.p2.x = cX - iOffs;

AddLine(l);

l.p1.y = cY - iOffs;

l.p2.y = cY - iOffs;

AddLine(l);

l.p1.x = cX + iOffs;

l.p2.x = cX + iOffs;

AddLine(l);

}

void Scene3D::UpdateDisplay()

{

int x;

if (pTV)

{

for(x = 0; x< iNumLines; x++)

{

pTV->draw\_line(xyztox(lines[x].p1.x,lines[x].p1.y,lines[x].p1.z),

xyztoy(lines[x].p1.x,lines[x].p1.y,lines[x].p1.z),

xyztox(lines[x].p2.x,lines[x].p2.y,lines[x].p2.z),

xyztoy(lines[x].p2.x,lines[x].p2.y,lines[x].p2.z),

1);

}

}

}

void Scene3D::AddSphere(float r, Point3D c)

{

// r = the radius of the sphere

// c = the centre point of the sphere

Line3D ln1;

int x;

int y;

float flat;

float flong;

float cr; // the radius of a circle inside the

float pi = 3.1416259;

for(y=-3; y<= 3; y++) // draw lines in circles parallel to the equator

{

flat = y\*pi/6;

ln1.p1.y = r\*sin(flat)+c.y;

ln1.p2.y = ln1.p1.y;

// calculated the y coordinate of a circle in the sphere

for(x=0; x <= 12; x++)

{

flong = x\*pi/6;

cr = r\*cos(flat);

ln1.p1 = ln1.p2;

ln1.p2.x = cr\*cos(flong) + c.x;

ln1.p2.z = cr\*sin(flong) + c.z;

if (x != 0)

{

AddLine(ln1);

}

}

for(x=0; x <= 12; x++) // reverse the wire pattern so it looks like a grid

{

// draw the pattern parallel to the meridians

flong = x\*pi/6;

for(y=-3; y<= 3; y++)

{

flat = y\*pi/6;

ln1.p1 = ln1.p2;

cr = r\*cos(flat);

ln1.p2.x = cr\*cos(flong)+c.x;

ln1.p2.z = cr\*sin(flong)+c.z;

ln1.p2.y = r\*sin(flat)+c.y;

// calculated the y coordinate of a circle in the sphere

if (y != -3)

{

AddLine(ln1);

}

}

}

}

}

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

NunchuckSensorsToTV.ino

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

#include <math.h>

#include <Wire.h>

#include "WiiChuck1.h"

#include <TVout.h>

#include <stdio.h>

//#include "notedefs.h" //note definitions for audio output(not used)

#include "lib3d.h"

#define SETUP\_WIDTH 120

#define WIDTH 119 // visible width...

#define HEIGHT 96

#define SOUND\_OUT\_PIN 2

#define MAX\_ROLL 15 // nunchuck max roll

#define MAX\_PITCH 90 // nunchuck max pitch

TVout cTV;

WiiChuck cChuck = WiiChuck();

Scene3D cScene;

int iLeftY = HEIGHT / 2;

int iRightY = HEIGHT / 2;

int angleStart, currentAngle;

int tillerStart = 0;

double angle;

#define CUBE 20

void setup()

{

cChuck.begin();

cChuck.update();

cChuck.calibrateJoy();

cTV.begin(\_PAL,SETUP\_WIDTH,HEIGHT); //for devices with only 1k sram(m168) use TV.begin(\_PAL,128,56)

cTV.clear\_screen();

cScene.CreateScene(WIDTH,HEIGHT,&cTV);

cScene.AddCube(0,0,0,CUBE);

}

void loop()

{

char buffer[20];

int iRoll;

float fRollAngle;

int iPitch;

float fPitchAngle;

cTV.delay\_frame(1);

cChuck.update();

cTV.clear\_screen();

iRoll = cChuck.readRoll();

fRollAngle = (PI \* iRoll)/MAX\_ROLL;

iPitch = cChuck.readPitch();

fPitchAngle = (PI \* iPitch)/MAX\_PITCH;

sprintf(buffer,"%d %d",iRoll,iPitch);

//cTV.print\_str(0,0,buffer); //deprecated

cTV.print(0,0,buffer); //Arduino 1.0

// cTV.draw\_line(0,iLeftY - iRoll, WIDTH, iLeftY + iRoll, 1);

cScene.SetHAngle(fRollAngle);

cScene.SetVAngle(fPitchAngle);

cScene.UpdateDisplay();

/\* Audio code not used SJW

if (iRoll > MAX\_ROLL)

{

cTV.tone( NOTE\_D4 );

}

else if (iRoll < -MAX\_ROLL)

{

cTV.tone( NOTE\_B4 );

}

else

{

cTV.noTone();

}

\*/

/\*

Serial.print(", ");

Serial.print(chuck.readPitch());

Serial.print(", ");

Serial.print((int)chuck.readAccelX());

Serial.print(", ");

Serial.print((int)chuck.readAccelY());

Serial.print(", ");

Serial.print((int)chuck.readAccelZ());

// Serial.print(", ");

// Serial.print((int)chuck.readJoyX());

// Serial.print(", ");

// Serial.print((int)chuck.readJoyY());

Serial.println();

\*/

}