THETA QUADRANT
ANIMATED 3D HOLOGRAPHIC LED DISPLAY

Ben Boldt

May 8, 2009

Department of Electrical and Computer Engineering
University of Minnesota Duluth
Duluth, MN 55812

Approved ______________________________ Date ______________
Advisor's Signature

TABLE OF CONTENTS

Abstract iii
I. Introduction 1
II. Description of Experiment 2
III. Procedure 5
IV. Results and Conclusions 16
V. References 17

Appendix A. Mathematica Calculations 18
Appendix B. Base Resistor Calculations 21
Appendix C. Schematic 23
Appendix D. Bitmap Image Program 26
Appendix E. Degree Readout Program 28
Appendix F. Animated 3D Line Program 33

LIST OF FIGURES

Figure 1. Inserting individual LEDs into wooden jig 6
Figure 2. Soldering across all LEDs to form permanent rows 6
Figure 3. Column soldering jig 7
Figure 4. Soldering columns in the jig 8
Figure 5. Box, bearings, aluminum disc, and Dragon12 board 8
Figure 6. LED display mounted in Lexan 9
Figure 7. Transistor banks complete 10
Figure 8. Optical sensors 11
Figure 9. Electrical contacts 12
Figure 10. First program, displaying a static image 13
Figure 11. Second program, reading out the angle of the display 13
Figure 12. 3D pyramid image 14
Figure 13. Belt drive 14

ABSTRACT

The purpose of this project is to design and build an apparatus capable of digitally displaying an animated 3D holographic image. This is accomplished by spinning a vertically mounted LED display about its edge in a circle, so as to sweep out a cylinder shape. Controlled by the familiar Dragon12 board used in ECE 2325 and utilizing some optical sensors, the display is refreshed and updated with a precision of 2 degrees giving a total resolution of 184,320 3D pixels.

The second purpose is the development of a program to draw straight lines through the curved path of the display. These lines can be used to draw any wire-frame 3D object. Basic 3D line, plane, and trigonometry techniques taught in MATH 1297 are used. Actual animation of floating 3D wire-frame objects is demonstrated.

I. INTRODUCTION

Theta Quadrant is a 3D holographic LED display. The project spins an LED display and sweeps out a virtual 3D image. The physical aspects of this project were time-consuming but lack relevance in the ECE field. Technology used includes optical sensors, a Dragon12 microcontroller board, and various transistors and digital logic chips. 3D line and plane equations are also used. A program runs on the microcontroller that makes calculations and refreshes the display. These calculations are based on input from sensors and all of the predefined line segments. The calculations are made and displayed quickly enough that we have an illusion of a complete 3D image.

II. DESCRIPTION OF EXPERIMENT

Theta Quadrant takes a database of endpoints and connects them using straight lines in the virtual cylindrical space created by spinning the display around. All that a programmer must do to draw a different image is to delete the existing endpoints and enter new ones. The program itself connects the points together with 3D lines.

Two 4-to-16 bit decoders are used to decode five column selector bits to 32 individual bits. Outputs from these decoders are connected to PNP transistors with a series 300Ω resistor. This is done to buffer the LEDs from the decoders. It greatly reduces the current drawn from the decoders and it also allows me to precisely control the voltage of the LEDs. The emitters of each of the 32 PNP transistors are tied together and powered by a variable voltage regulator set to the maximum voltage of the LEDs, which is 2.8 volts. There is also an option to increase the voltage to 5V while the display is refreshing, which produces a safe surge current of 142 mA.

As each column is selected, data is fed to each LED in that column directly from output bits of the microcontroller. These are also buffered using NPN transistors. The emitters of each of the 32 NPN transistors are tied directly to ground potential.

The brain of Theta Quadrant is an HC12-type microcontroller, specifically the MC9S12DP256. At first, a program was developed that would simply display a static, bitmap image on the display. This was done to test the display, and also to develop the part of the program that refreshes the display. The same code was reused in all subsequent programs. Next, another program was developed to test the optical degree sensor and programmatically changing the image that is shown on the display. Bitmap Arabic numbers were stored in memory, and the program copied the appropriate numbers to the appropriate areas of the display based on interrupts generated by the optical sensors. Finally, this program was expanded to become the program that could draw trigonometrically calculated straight lines across the curved path of the display, given only endpoints.

The program connects these lines by lighting up appropriate pixels as the display is spinning. The program spends most of its time just refreshing the display from an internal bitmap database. This database always contains exactly one frame of the data that is to be displayed for two degrees of rotation of the display. Each time the display moves an additional two degrees, an optical sensor is triggered. This changes the logic level of an external interrupt input pin of the microcontroller via a Schmitt trigger. This then invokes a special part of the program to handle the interrupt, called an interrupt service subroutine.

The interrupt service subroutine blanks the LED display and the display database. It then proceeds to look at each line and plots it into the display database properly by using equations derived in Mathematica that can produce straight lines along the curved path of the display. It is possible for a line to intersect the display as a point or as a line, so each method is chosen appropriately. Once the subroutine finishes going through each line and plotting the appropriate points and lines into the database, the execution returns to the normal, refreshing part of the program that refreshes the database onto the actual LED display. By design, refreshing picks back up where it left off, which could be at any point on the display. This produces a more consistent image.

III. PROCEDURE

The first semester of this senior design project was spent with my partner, David Buszmann. It was planned that I would create the apparatus during the first semester and that he would implement the software during the second semester. During the first semester, we met with Dr. Chris Carroll on a weekly basis for updates on my own progress. We also regularly looked at the feasibility of the project and where we were standing. This was helpful to keep us on schedule.

During the first semester, I purchased most of the materials and got working on the LED display. I first built an 8x8 pixel LED display to demonstrate my soldering skills and also to demonstrate a small-scale functional interface between the LEDs and the microcontroller. This was successful. From there, I developed a way of fabricating a much larger 32x32 pixel display, consisting of 1,024 LEDs. As shown in figure 1, I used a piece of wood with a small notch cut out of it and a piece of scotch tape over the notch. I then inserted each of 32 LEDs in between the wood and the tape with a tweezers. I then took one of the wires of the pre-stripped ribbon and soldered it all the way across all of the LEDs, as shown in Figure 2. This process was repeated for all 32 rows.

Figure 1. Inserting individual LEDs into wooden jig.

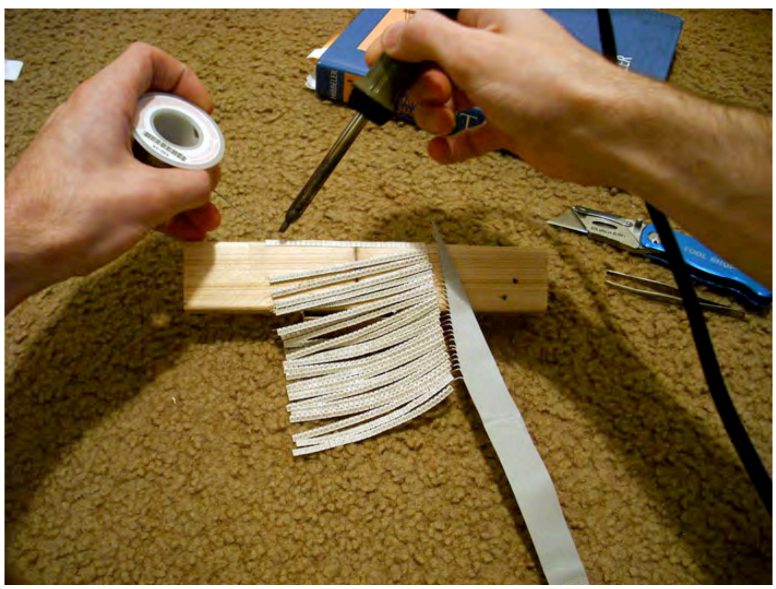


Figure 2. Soldering across all LEDs to form permanent rows.

Once all of the rows were done, a new jig was required for me to insert all of the rows and solder another ribbon to the other sides of the LEDs. These form columns. As seen in figure 3, the jig was made from cardboard, a laser-printed template, and 1/32” thickness birch veneer strips.

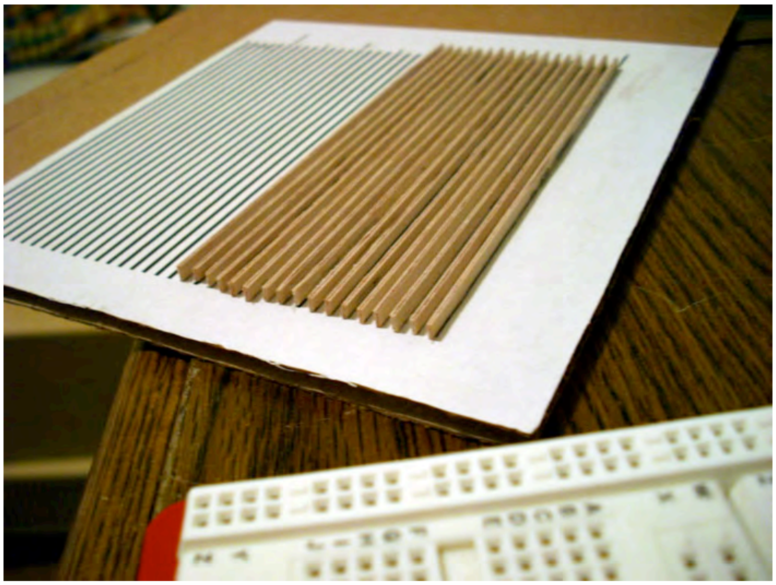


Figure 3. Column soldering jig.

Each strip had to be glued and set in place for a period of time until the glue could dry. Because of the small size, I was only able to glue one strip at a time, so this process took several days. Once the jig was complete, I inserted each row of the display and began to solder the columns together as seen in figure 4.

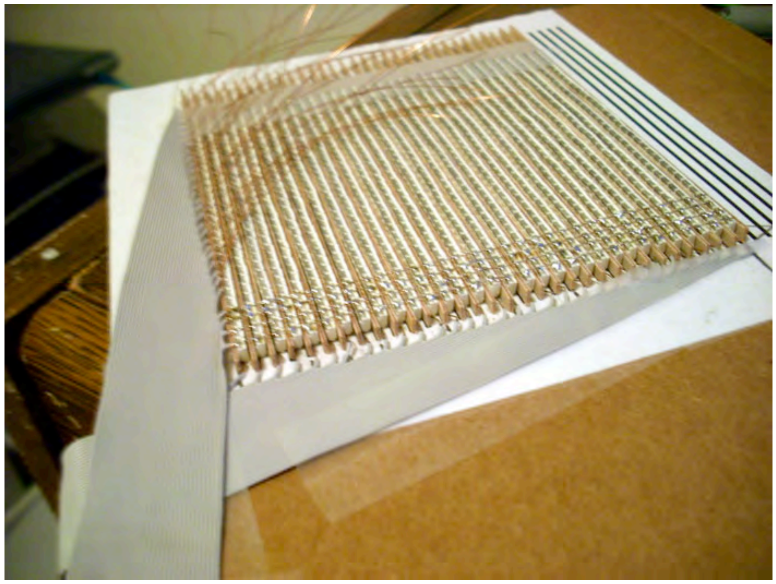


Figure 4. Soldering columns in the jig.

Also during the first semester, I was able to build a box and attach an aluminum disc with bearings. The Dragon12 board was attached directly to this aluminum disc. See figure 5.

Figure 5. Box, bearings, aluminum disc, and Dragon12 board.

This is all of the work that I got done during the first semester. At the beginning of the second semester, I built a housing for the display out of Lexan, which is similar to plexiglass. Balsa wood was used to accommodate a consistent space for the sandwiched display. Silicone glue was used to hold the parts together. See figure 6.

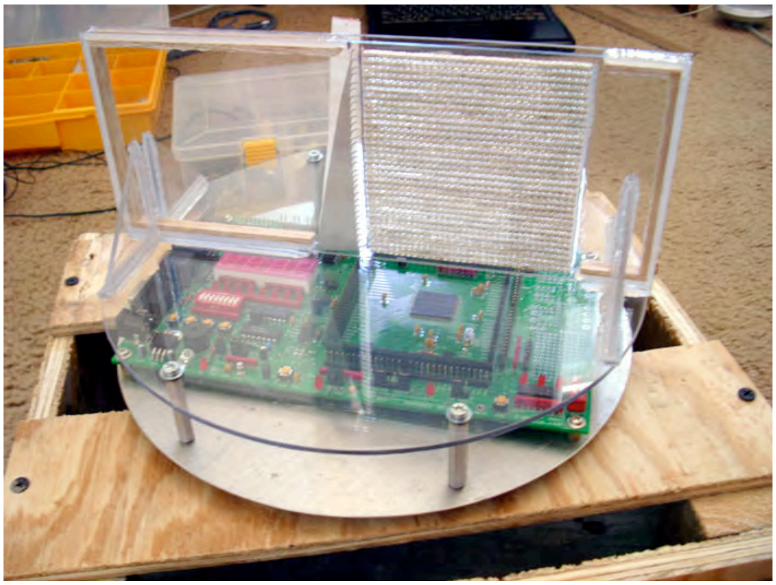


Figure 6. LED display mounted in Lexan.

Next, I needed an electrical connection between the display and the microcontroller board. To do that, I used two banks of transistors, one for each row and one for each column. Transistors that control the rows are connected directly to output pins of the microcontroller, and transistors that control the columns are connected to decoders, which are connected to just 5 output pins of the microcontroller. See figure 7.

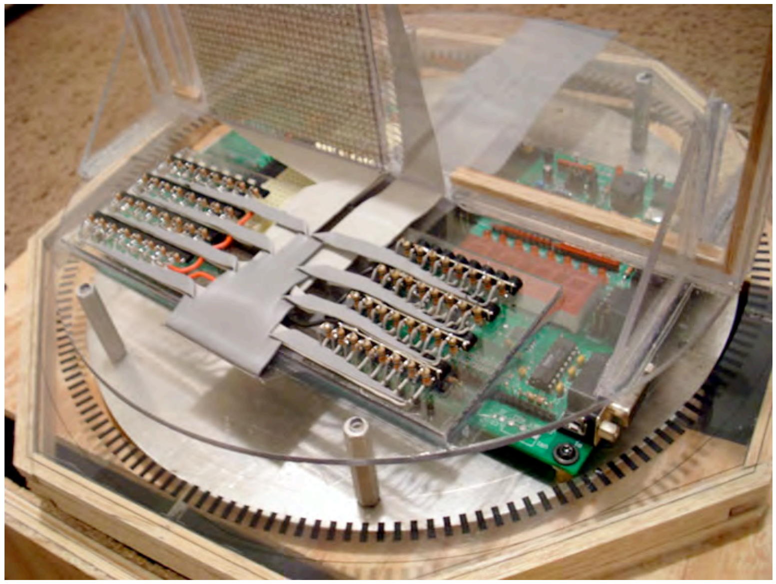


Figure 7. Transistor banks complete.

At about the same time, I was working on the optical sensors. A printed transparency, which can be seen in figure 7, was used to interrupt the sensors. This gives the microcontroller the ability to count these interruptions and, thus, know what angle the display is at. It also provides the ability for the display to not spin at a synchronous speed to display an image. Figure 8 shows the actual sensor better.

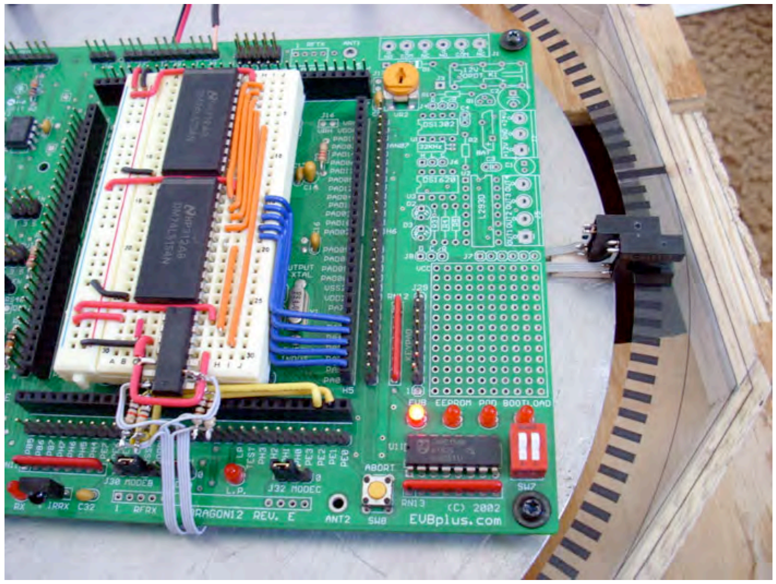


Figure 8. Optical sensors.

As you can see, there are two sensors and one of the lines is longer than the rest. This is to establish the home position of the display. A Schmitt trigger is used to ensure interrupts are called only once per line on the transparent sheet.

Next I needed a brushing system that could get power onto the moving disc. That was accomplished by taking some copper and PVC fittings and contacting them with some spring-loaded steel wire. It's a bit more difficult to explain that to just look at, so see figure 9.

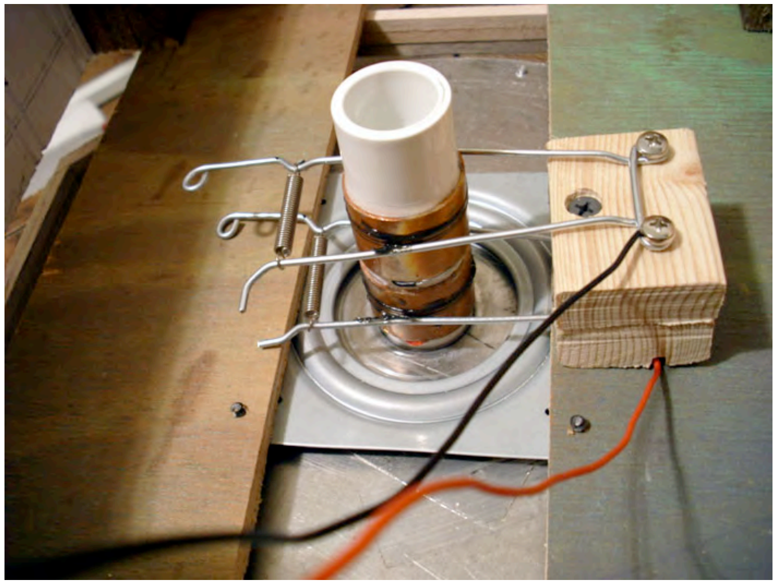


Figure 9. Electrical contacts.

It's a little rudimentary, but it works. Power on the board is unstable due to vibrations and other inconsistencies in the contacts, so large capacitors and voltage regulators are located on the disc to stabilize the power.

Now that power is on the disc and the display is connected to the microcontroller, it is now possible to begin writing software. So I wrote some test programs. The first program (figure 10) simply put a bitmap image on the display. This is significant because the same code that refreshes the display in this program is used in each subsequent program. The second program (figure 11) utilizes the optical sensors and implements a display database. Bitmaps of numerals are copied into the display database during interrupts, and upon returning, are refreshed onto the actual LED display. This programming was also reused in the final program.

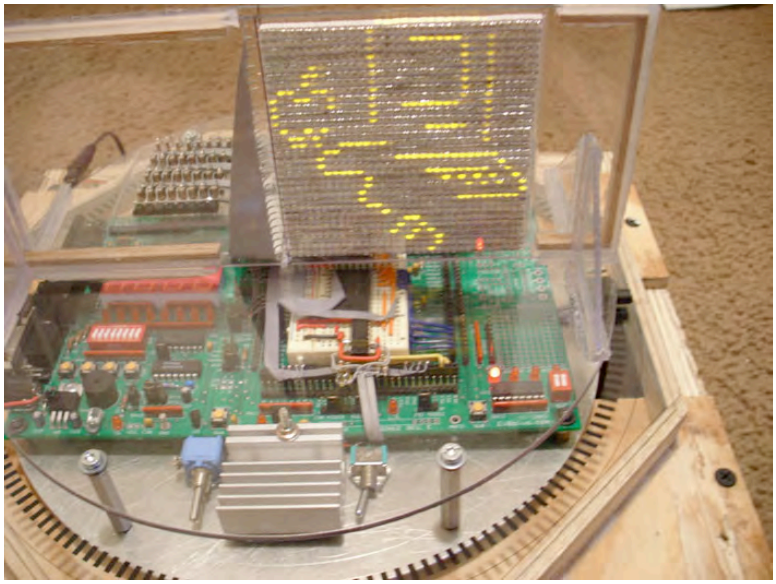


Figure 10. First program, displaying a static image.

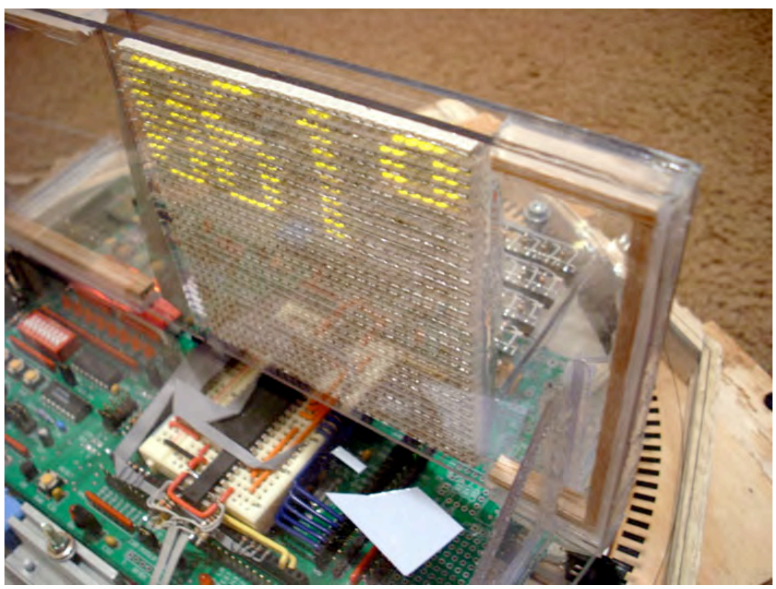


Figure 11. Second program, reading out the angle of the display.

It was at this point that I needed to figure out equations for making straight lines along the curved path of the display. I used Mathematica for this and found the equations fairly easily. I then programmed these equations into the microcontroller in such a way that simply entering endpoints would result in the program connecting them together in virtual 3D space. See figure 12 for a 3D pyramid image produced by this program.

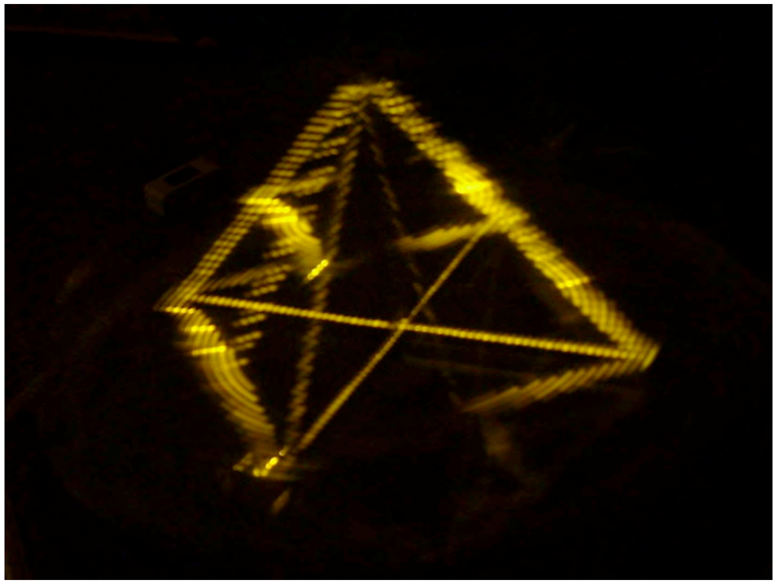


Figure 12. 3D pyramid image.

Once that was complete, I proceeded to hook a motor up to this thing so that I would not have to spin it by hand. You can see the motor in figure 13.

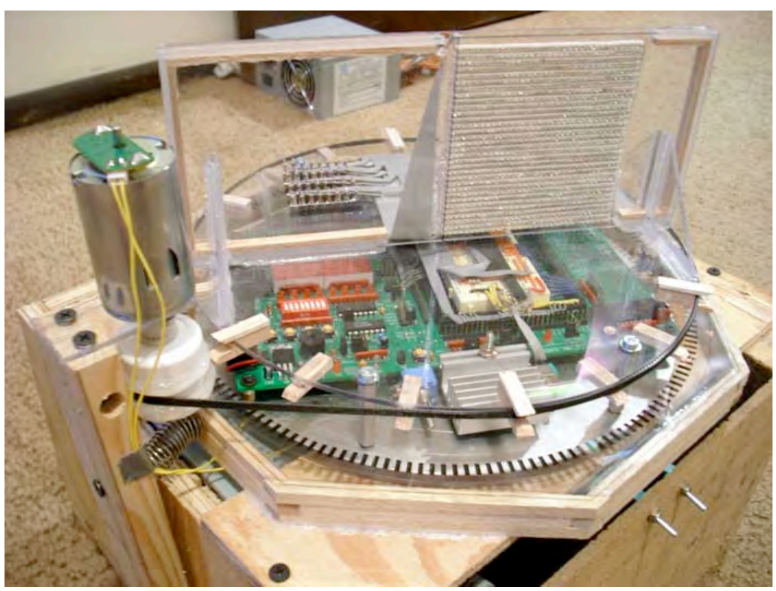


Figure 13. Belt drive.

One last detail is that I added a small triangular prism in the corner of the display that moves up and down. This allows me to demonstrate 3D animation. That was the last bit of work that I did on this project before the presentation and writing the report.

IV. RESULTS AND CONCLUSIONS

Theta Quadrant was able to operate properly with all three programs. In the first program, a static image was successfully displayed on the LED display. In the second program, the angle of the display was successfully read out onto the display by copying appropriate bitmap images from a database into the display database. Then upon returning from the copying process, the image was refreshed onto the display as expected. The final program was able to draw straight lines across the curved path of the display. Any straight line at any length or angle that you can imagine drawing inside of the virtual cylinder area is possible by simply entering the appropriate endpoints into the program. The program automatically connects the points together by lighting up LEDs. Basic animation was also introduced to demonstrate future potential of animation. One object in the 3D field remains stationary while another object moves up and down.

The results are exactly as planned, except for a ghosting effect on the display. This is caused by the storage effect of the inexpensive transistors I used. This can easily be corrected by replacing with better transistors.

Overall, the project is a great success!

V. REFERENCES

I have no references to any literature or Internet web sites. All knowledge used came form the following UMD classes:

- ECE 1315
- ECE 2325
- MATH 1297
- MATH 3280

Dr. Carroll was my project advisor and he gave many helpful tips and put things in perspective on a weekly basis.

APPENDIX A

MATHEMATICA CALCULATIONS

Theta Quadrant r & Z Calculation

Equation of the display plane, with respect to θ.

$$\vec{n} == [\mathrm{Sin}[\theta], \mathrm{Cos}[\theta], 0]$$
$$p_0 == (0, 0, 0)$$

$$x\,\mathrm{Sin}[\theta] + y\,\mathrm{Cos}[\theta] == 0$$

Parametric equations of the 3D line in rectangular coordinates.

$$x == x_{start} + (x_{end} - x_{start})\ t$$
$$y == y_{start} + (y_{end} - y_{start})\ t$$
$$z == z_{start} + (z_{end} - z_{start})\ t$$

Combining the plane and line equations together, solving for t.

$$\mathrm{Solve}[(x_{start} + (x_{end} - x_{start})\ t)\ \mathrm{Sin}[\theta] + (y_{start} + (y_{end} - y_{start})\ t)\ \mathrm{Cos}[\theta] == 0,\ t]$$

$$\left\{\left\{t \to \frac{-\mathrm{Sin}[\theta]\ x_{start} - \mathrm{Cos}[\theta]\ y_{start}}{\mathrm{Sin}[\theta]\ x_{end} - \mathrm{Sin}[\theta]\ x_{start} + \mathrm{Cos}[\theta]\ y_{end} - \mathrm{Cos}[\theta]\ y_{start}}\right\}\right\}$$

Inserting t to find point (x,y,z) at θ.

$$x == x_{start} + (x_{end} - x_{start}) * \frac{-\mathrm{Sin}[\theta]\ x_{start} - \mathrm{Cos}[\theta]\ y_{start}}{\mathrm{Sin}[\theta]\ x_{end} - \mathrm{Sin}[\theta]\ x_{start} + \mathrm{Cos}[\theta]\ y_{end} - \mathrm{Cos}[\theta]\ y_{start}}$$
$$y == y_{start} + (y_{end} - y_{start}) * \frac{-\mathrm{Sin}[\theta]\ x_{start} - \mathrm{Cos}[\theta]\ y_{start}}{\mathrm{Sin}[\theta]\ x_{end} - \mathrm{Sin}[\theta]\ x_{start} + \mathrm{Cos}[\theta]\ y_{end} - \mathrm{Cos}[\theta]\ y_{start}}$$
$$z == z_{start} + (z_{end} - z_{start}) * \frac{-\mathrm{Sin}[\theta]\ x_{start} - \mathrm{Cos}[\theta]\ y_{start}}{\mathrm{Sin}[\theta]\ x_{end} - \mathrm{Sin}[\theta]\ x_{start} + \mathrm{Cos}[\theta]\ y_{end} - \mathrm{Cos}[\theta]\ y_{start}}$$

$$x == x_{start} + \frac{(x_{end} - x_{start})\ (-\mathrm{Sin}[\theta]\ x_{start} - \mathrm{Cos}[\theta]\ y_{start})}{\mathrm{Sin}[\theta]\ x_{end} - \mathrm{Sin}[\theta]\ x_{start} + \mathrm{Cos}[\theta]\ y_{end} - \mathrm{Cos}[\theta]\ y_{start}}$$

$$y == y_{start} + \frac{(y_{end} - y_{start})\ (-\mathrm{Sin}[\theta]\ x_{start} - \mathrm{Cos}[\theta]\ y_{start})}{\mathrm{Sin}[\theta]\ x_{end} - \mathrm{Sin}[\theta]\ x_{start} + \mathrm{Cos}[\theta]\ y_{end} - \mathrm{Cos}[\theta]\ y_{start}}$$

$$z == \frac{(-\mathrm{Sin}[\theta]\ x_{start} - \mathrm{Cos}[\theta]\ y_{start})\ (z_{end} - z_{start})}{\mathrm{Sin}[\theta]\ x_{end} - \mathrm{Sin}[\theta]\ x_{start} + \mathrm{Cos}[\theta]\ y_{end} - \mathrm{Cos}[\theta]\ y_{start}} + z_{start}$$

Switching to cylindrical system.
Input: $x = r\,\mathrm{Cos}[\theta]$. $y = r\,\mathrm{Sin}[\theta]$. $z = z$.
Output: $r = \frac{y}{\mathrm{Sin}[\theta]}$. $z = z$.

$$x_{start} := r_{start}\,\mathrm{Cos}[\theta_{start}]$$
$$x_{end} := r_{end}\,\mathrm{Cos}[\theta_{end}]$$
$$y_{start} := r_{start}\,\mathrm{Sin}[\theta_{start}]$$
$$y_{end} := r_{end}\,\mathrm{Sin}[\theta_{end}]$$
$$z_{start} := z_{start}$$
$$z_{end} := z_{end}$$

$$\mathrm{FullSimplify}\left[r == \frac{y_{start} + \frac{(y_{end}-y_{start})\,(-\mathrm{Sin}[\theta]\,x_{start}-\mathrm{Cos}[\theta]\,y_{start})}{\mathrm{Sin}[\theta]\,x_{end}-\mathrm{Sin}[\theta]\,x_{start}+\mathrm{Cos}[\theta]\,y_{end}-\mathrm{Cos}[\theta]\,y_{start}}}{\mathrm{Sin}[\theta]}\right]$$

$$\mathrm{FullSimplify}\left[z == \frac{(\mathrm{Sin}[\theta]\,x_{start} + \mathrm{Cos}[\theta]\,y_{start})\,(z_{end} - z_{start})}{\mathrm{Sin}[\theta]\,(-x_{end} + x_{start}) + \mathrm{Cos}[\theta]\,(-y_{end} + y_{start})} + z_{start}\right]$$

$$r == -\frac{\mathrm{Sin}[\theta_{end} - \theta_{start}]\,r_{end}\,r_{start}}{\mathrm{Sin}[\theta + \theta_{end}]\,r_{end} - \mathrm{Sin}[\theta + \theta_{start}]\,r_{start}}$$

$$z + \frac{\mathrm{Sin}[\theta + \theta_{end}]\,r_{end}\,(z_{end} - z_{start})}{\mathrm{Sin}[\theta + \theta_{end}]\,r_{end} - \mathrm{Sin}[\theta + \theta_{start}]\,r_{start}} == z_{end}$$

Notice the exclusive use of the sine function.

Extracting parts that are independent of θ and thus only need to be calculated and stored when the program is first run.

$$\mathrm{NUM} := \mathrm{Sin}[\theta_{end} - \theta_{start}]\,r_{end}\,r_{start}$$
$$\Delta Z := z_{end} - z_{start}$$

Extracting parts that are used multiple times during on-the-fly calculation. Trig modification to ensure $0° \leq a \leq 360°$ in Sin[a].

$$\mathrm{V1} := \mathrm{Sin}[\theta - \theta_{end} + 360\,°]\,r_{end}$$
$$\mathrm{V2} := \mathrm{Sin}[\theta - \theta_{start}]\,r_{start} - \mathrm{V1}$$

Using these new variables to find r and z.

$$r == \frac{\mathrm{NUM}}{\mathrm{V2}}$$

$$z == z_{end} + \frac{\mathrm{V1} * \Delta Z}{\mathrm{V2}}$$

$$r == \frac{\mathrm{Sin}[\theta_{end} - \theta_{start}]\,r_{end}\,r_{start}}{-\mathrm{Sin}[\theta - \theta_{end}]\,r_{end} + \mathrm{Sin}[\theta - \theta_{start}]\,r_{start}}$$

$$z == z_{end} + \frac{\mathrm{Sin}[\theta - \theta_{end}]\,r_{end}\,(z_{end} - z_{start})}{-\mathrm{Sin}[\theta - \theta_{end}]\,r_{end} + \mathrm{Sin}[\theta - \theta_{start}]\,r_{start}}$$

APPENDIX B

BASE RESISTOR CALCULATIONS

NPN Transistor Calculations:

Known values:
Maximum current in/out of a microcontroller pin = 25mA
Maximum load current = Collector Current (I_C) = 30mA

Transistor's minimum current gain (h_{FE}):

$$h_{FE(\min)} > 5 \times \frac{I_C}{I_{Pin(\max)}}$$

$$h_{FE(\min)} > 5 \times \frac{.030A}{.025A}$$

$$h_{FE(\min)} > 6$$

I chose the 2N3904 transistor for 3 reasons:
- The current gain (h_{FE}) is at least 60.
    - 60 > 6
- Maximum sustained collector current (I_C) is 200 mA
    - 200 mA > 30mA
- I already had some that I could use.

Base Resistance Calculation:
The LED voltage may be increased by several volts in order to take advantage of the 500mA surge current capability of the LEDs I used. It will definitely never exceed 8V, so I chose to use this value.

$$R_B = \frac{V_C \times h_{FE}}{5 \times I_C}$$

$$R_B = \frac{8V \times 6}{5 \times .03A}$$

$$R_B = 320\Omega$$

PNP Transistor Calculations:

Known values:
Maximum current in/out of a microcontroller pin = 25mA
Maximum load current = Collector Current (I_C) = 960mA

Transistor's minimum current gain (h_{FE}):

$$h_{FE(\min)} > 5 \times \frac{I_C}{I_{Pin(\max)}}$$

$$h_{FE(\min)} > 5 \times \frac{.960A}{.025A}$$

$$h_{FE(\min)} > 192$$

I chose the BCP53T1 transistor for 3 reasons:
- The current gain (h_{FE}) is at least 192.
    - 250 > 192
- Maximum sustained collector current (I_C) is 1500 mA
    - 1500 mA > 960mA
- They were very inexpensive on eBay.

Base Resistance Calculation:
The LED voltage may be increased by several volts in order to take advantage of the 500mA surge current capability of the LEDs I used. It will definitely never exceed 8V, so I chose to use this value.

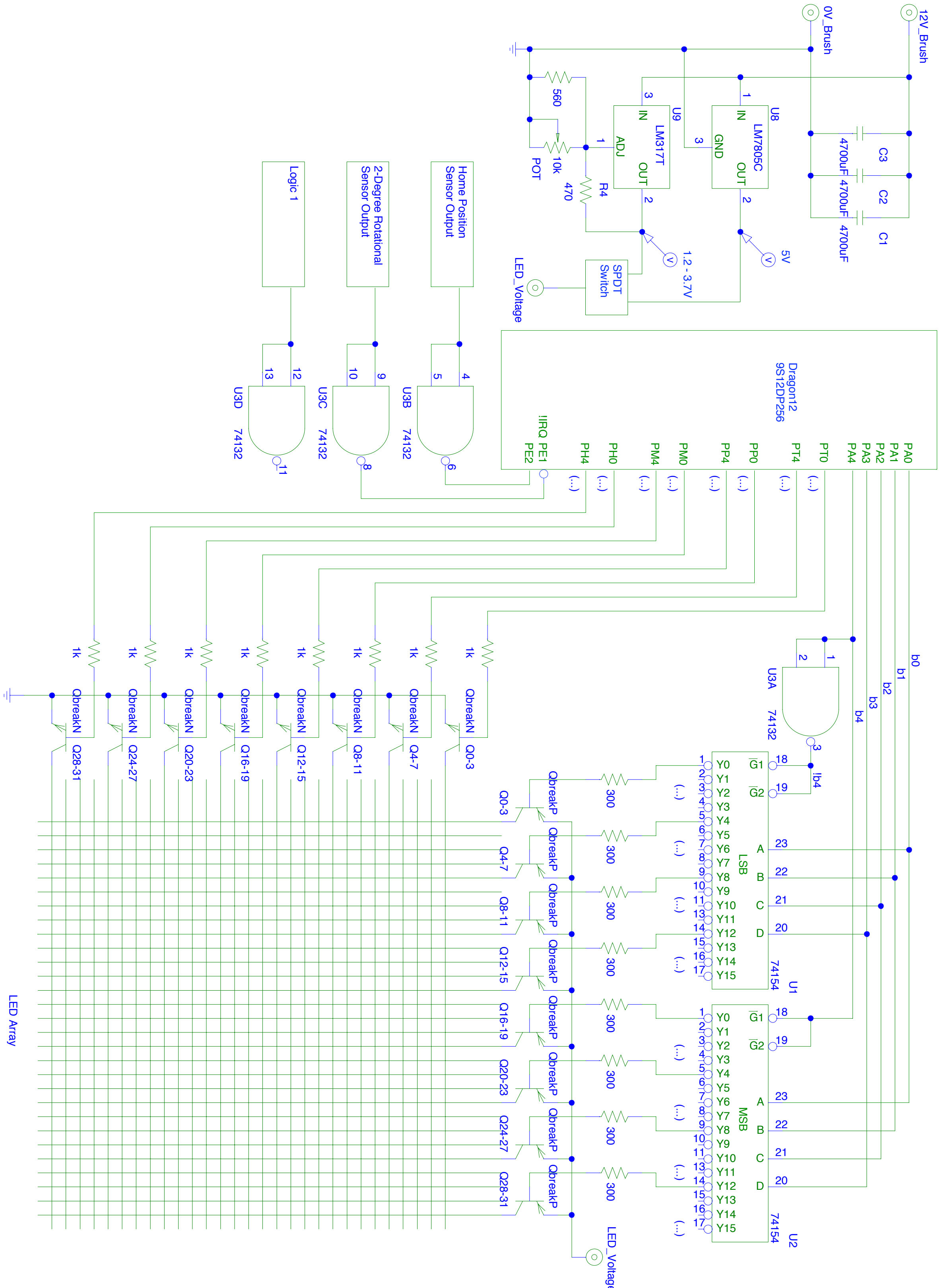
$$R_B = \frac{V_C \times h_{FE}}{5 \times I_C}$$

$$R_B = \frac{8V \times 192}{5 \times .96A}$$

$$R_B = 320\Omega$$

APPENDIX C

SCHEMATIC


0V_Brush
12V_Brush
U8
LM7805C
IN
GND
OUT
U9
LM317T
IN
ADJ
OUT
560
10k
POT
R4
470
C3
C2
C1
4700uF
4700uF
4700uF
5V
1.2 - 3.7V
SPDT
Switch
LED_Voltage
Home Position
Sensor Output
2-Degree Rotational
Sensor Output
Logic 1
U3B
U3C
U3D
U3A
74132
Dragon12
9S12DP256
!IRQ PE1
PE2
PH4
PH0
PM4
PM0
PP4
PP0
PT4
PT0
PA0
PA1
PA2
PA3
PA4
(...)
b0
b1
b2
b3
b4
!b4
1k
QbreakN
Q0-3
Q4-7
Q8-11
Q12-15
Q16-19
Q20-23
Q24-27
Q28-31
QbreakP
300
U1
74154
LSB
U2
74154
MSB
G1
G2
A
B
C
D
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7
Y8
Y9
Y10
Y11
Y12
Y13
Y14
Y15
LED Array

APPENDIX D

BITMAP IMAGE PROGRAM

```
*** Equate Zone ***

PortA        equ $0000
PortH        equ $0260
PortM        equ $0250
PortP        equ $0258
PortT        equ $0240

DDRA         equ $0002
DDRH         equ $0262
DDRM         equ $0252
DDRP         equ $025A
DDRT         equ $0242

*** Data & Subroutine Zone ***

             org $1000

Image        fdb $0000, $0600
             fdb $0000, $0900
             fdb $0000, $1100
             fdb $0000, $2500
             fdb $0000, $4900
             fdb $3FFF, $5500
             fdb $0000, $5900
             fdb $8000, $5500
             fdb $9FC4, $5900
             fdb $A024, $5000
             fdb $A024, $5800
             fdb $A024, $5006
             fdb $A024, $5009
             fdb $A020, $4010
             fdb $A020, $4028
             fdb $8020, $4024
             fdb $8000, $4010
             fdb $8000, $0008
             fdb $8001, $0040
             fdb $7FF9, $0080
             fdb $0001, $0C80
             fdb $0001, $1300
             fdb $0001, $1000
             fdb $0001, $1000
             fdb $0060, $9000
             fdb $0084, $9000
             fdb $0682, $6000
             fdb $0344, $0000
             fdb $0042, $0000
             fdb $0082, $0000
             fdb $0084, $0000
             fdb $0078, $0000

*** Main Program Zone ***

             org $2000

             BSET DDRA,$FF
             BSET DDRT,$FF
             BSET DDRP,$FF
             BSET DDRM,$FF
             BSET DDRH,$FF
             LDS #$1FFF

             BSET PortA,$FF
             CLR PortT
             CLR PortP
             CLR PortM
             CLR PortH

ScanHome     LDX #Image
NextCol      INC PortA

             LDD 2,X+
             STAA PortH
             STAB PortM

             LDD 2,X+
             STAA PortP
             STAB PortT

             CLR PortH
             CLR PortM
             CLR PortP
             CLR PortT

             CPX #Image+128
             BEQ ScanHome
             BRA NextCol
```

APPENDIX E

DEGREE READOUT PROGRAM

```
*** Equate Zone ***

PortA        equ $0000
PortH        equ $0260
PortM        equ $0250
PortP        equ $0258
PortT        equ $0240

DDRA         equ $0002
DDRH         equ $0262
DDRM         equ $0252
DDRP         equ $025A
DDRT         equ $0242

PortE        EQU $0008

IRQVector    EQU $3E72    ; External Interrupt Request Vector (Port E bit 1)
XIRQVector   EQU $3E74    ; Nonmaskable External Interrupt Request Vector (Port E bit 0)
IntCR        EQU $001E    ; Interrupt Control Register

*** Data & Subroutine Zone ***

             org $1000

IRQ_ISS:     BRCLR PortE,#%00000100,Increment        ; Checks if the display is at home position.

             CLR Counter
             CLR Counter+1
             CLR Counter+2
             BRA Update

Increment    LDAA Counter
             CMPA #$08
             BGE Tens
             INC Counter
             INC Counter
             BRA Update

Tens         CLR Counter
             LDAA Counter+1
             CMPA #$09
             BGE Hundreds
             INC Counter+1
             BRA Update

Hundreds     CLR Counter+1
             LDAA Counter+2
             INC Counter+2
             CMPA #$03
             BLT Update
             CLR Counter+2

Update       LDAB #$00
             JSR Point                  ; Points X at correct image data.
             LDY #Image+32 ; Points Y at display database
             LDAB #$8
Copy1        LDAA 1,X+                  ; Loads data, increments
             STAA 1,Y+                  ; Loads data, increments
             LDAA 1,X+
             STAA 3,Y+
             DBNE B,Copy1

             LDAB #$01
             JSR Point
             LDY #Image+64 ; Points Y at display database
             LDAB #$8
Copy2        LDAA 1,X+                  ; Loads data, increments
             STAA 1,Y+                  ; Loads data, increments
             LDAA 1,X+
             STAA 3,Y+
             DBNE B,Copy2

             LDAB #$02
             JSR Point
             LDY #Image+96 ; Points Y at display database
             LDAB #$8
Copy3        LDAA 1,X+                  ; Loads data, increments
             STAA 1,Y+                  ; Loads data, increments
             LDAA 1,X+
             STAA 3,Y+
             DBNE B,Copy3

             LDX #ndeg
             LDY #Image
             LDAB #$8
Copy4        LDAA 1,X+                  ; Loads data, increments
             STAA 1,Y+                  ; Loads data, increments
             LDAA 1,X+
             STAA 3,Y+
```

```
                DBNE B,Copy4
Exit            RTI

Point:          LDX #Counter
                LDAA B,X
                CMPA #$00
                BNE t1
                LDX #n0
                RTS
t1              CMPA #$01
                BNE t2
                LDX #n1
                RTS
t2              CMPA #$02
                BNE t3
                LDX #n2
                RTS
t3              CMPA #$03
                BNE t4
                LDX #n3
                RTS
t4              CMPA #$04
                BNE t5
                LDX #n4
                RTS
t5              CMPA #$05
                BNE t6
                LDX #n5
                RTS
t6              CMPA #$06
                BNE t7
                LDX #n6
                RTS
t7              CMPA #$07
                BNE t8
                LDX #n7
                RTS
t8              CMPA #$08
                BNE t9
                LDX #n8
                RTS
t9              LDX #n9
                RTS


Counter         fcb $00, $00, $00


Image           fdb $0000, $0000
                fdb $0000, $0000
                fdb $0000, $0000
                fdb $0000, $0000
                fdb $0000, $0000
                fdb $0000, $0000
                fdb $0000, $0000
                fdb $0000, $0000
                fdb $0000, $0000
                fdb $0000, $0000
                fdb $0000, $0000
                fdb $0000, $0000
                fdb $0000, $0000
                fdb $0000, $0000
                fdb $0000, $0000
                fdb $0000, $0000
                fdb $0000, $0000
                fdb $0000, $0000
                fdb $0000, $0000
                fdb $0000, $0000
                fdb $0000, $0000
                fdb $0000, $0000
                fdb $0000, $0000
                fdb $0000, $0000
                fdb $0000, $0000
                fdb $0000, $0000
                fdb $0000, $0000
                fdb $0000, $0000
                fdb $0000, $0000
                fdb $0000, $0000
                fdb $0000, $0000
                fdb $0000, $0000

n0              fdb $0000
                fdb $3FE0
                fdb $7FF0
                fdb $4C10
                fdb $4210
                fdb $4190
                fdb $7FF0
```

```
                fdb $3FE0

n1              fdb $0000
                fdb $0000
                fdb $0000
                fdb $7FF0
                fdb $7FF0
                fdb $2000
                fdb $0000
                fdb $0000

n2              fdb $0000
                fdb $3810
                fdb $7E10
                fdb $4710
                fdb $4110
                fdb $40D0
                fdb $40F0
                fdb $2030

n3              fdb $0000
                fdb $43E0
                fdb $67F0
                fdb $5410
                fdb $5410
                fdb $4C10
                fdb $4410
                fdb $4020

n4              fdb $0100
                fdb $7FF0
                fdb $7FF0
                fdb $2100
                fdb $1100
                fdb $0900
                fdb $0500
                fdb $0300

n5              fdb $0000
                fdb $43E0
                fdb $47F0
                fdb $4410
                fdb $4410
                fdb $4410
                fdb $7C10
                fdb $7C20

n6              fdb $0000
                fdb $03E0
                fdb $47F0
                fdb $4410
                fdb $4410
                fdb $6610
                fdb $3FF0
                fdb $1FE0

n7              fdb $0000
                fdb $7800
                fdb $7E00
                fdb $4FF0
                fdb $41F0
                fdb $4000
                fdb $4000
                fdb $4000

n8              fdb $0000
                fdb $3DE0
                fdb $7DF0
                fdb $4210
                fdb $4210
                fdb $4210
                fdb $7DF0
                fdb $3DE0

n9              fdb $0000
                fdb $3FC0
                fdb $7FE0
                fdb $4330
                fdb $4110
                fdb $4110
                fdb $7F10
                fdb $3E00

ndeg            fdb $0000
                fdb $3800
                fdb $7C00
                fdb $4400
                fdb $4400
                fdb $7C00
```

```
                fdb $3800
                fdb $0000

*** Main Program Zone ***

                org $2000

                LDS #$1FFF

                BSET DDRA,$FF
                BSET DDRT,$FF
                BSET DDRP,$FF
                BSET DDRM,$FF
                BSET DDRH,$FF

                BSET PortA,$FF
                CLR PortT
                CLR PortP
                CLR PortM
                CLR PortH

                BSET IntCR,#$80            ; Set !IRQ to be triggered on falling edge.

                LDD #IRQ_ISS  ; } Connect IRQVector to IRQ_ISS subroutine
                STD IRQVector ; }

                CLI                        ; Enable Interrupts

ScanHome        LDX #Image
NextCol         INC PortA

                LDD 2,X+
                STAA PortH
                STAB PortM

                LDD 2,X+
                STAA PortP
                STAB PortT

                CLR PortH
                CLR PortM
                CLR PortP
                CLR PortT

                CPX #Image+128
                BEQ ScanHome
                BRA NextCol
```

APPENDIX F

ANIMATED 3D LINE PROGRAM

```
*** Equate Zone ***

PortA         equ $0000
PortH         equ $0260
PortM         equ $0250
PortP         equ $0258
PortT         equ $0240

DDRA          equ $0002
DDRH          equ $0262
DDRM          equ $0252
DDRP          equ $025A
DDRT          equ $0242

PortE         EQU $0008

IRQVector     EQU $3E72     ; External Interrupt Request Vector (Port E bit 1)
XIRQVector    EQU $3E74     ; Nonmaskable External Interrupt Request Vector (Port E bit 0)
IntCR         EQU $001E     ; Interrupt Control Register


;*** Data & Subroutine Zone ***

              org $1000

;*** START IRQ ISS ***
IRQ_ISS:      CLR PortT     ; }
              CLR PortP     ; } clear the actual display (prevents burn)
              CLR PortM     ; }
              CLR PortH     ; }
              JSR Clear     ; clear the display database
              BRCLR PortE,#%00000100,Increment         ; Checks if the display is at home position.

              ;* Start Home position *
              LDX #$166                   ; Spin Display Clockwise
              STX Degree
              ;CLR Degree   ; Spin Display Counter-Clockwise
              ;CLR Degree+1

              ; Start Animation
              LDAA AnimCnt
              BNE AnimNext
              ; Change direction of animation
              COM AnimDirec
              LDAA #15
              STAA AnimCnt
AnimNext      LDAA AnimDirec
              BNE AnimDown

                ; Animate Up
              LDAA #9
              LDX #Animated+1
AnimLoop      INC 6,X+
              INC 0,X
              LDAB #10
              ABX
              DECA
              BNE AnimLoop
              DEC AnimCnt
              BRA Update

                ; Animate Down
AnimDown      LDAA #9
              LDX #Animated+1
AnimLoop2     DEC 6,X+
              DEC 0,X
              LDAB #10
              ABX
              DECA
              BNE AnimLoop2
              DEC AnimCnt
              BRA Update
              ; End Animation

              ;* End Home Position

Increment     LDX Degree
              DEX           ; Spin Display Clockwise
              DEX
              ;INX          ; Spin Display Counter-Clockwise
              ;INX
              STX Degree

Update        LDX #Points
              LDAA PointCnt
NextLine      PSHA          ; PUSH: Line Counter

              LDY 2,X       ;start        ; }
              CPY Degree                  ; }
```

```
            LBGT LineDone ; } Determine if this line is even drawn at this angle of the display
            LDY 8,X                    ; }
            CPY Degree                 ; } If not, proceed to next line.
            LBLT LineDone ; }

            CPY 2,X                    ; Check to see if the intersection is a line
            LBNE Intersect             ; If not, branch to find a point intersection

; *** CALCULATION: LINE-DISPLAY INTERSECTION IS A LINE ***
; * Determine if this is a vertical line *
            LDY 0,X
            CPY 6,X
            BNE TestHoriz ; Branch if NOT vertical.

            EXG Y,B
            ASLB                       ; multiply B times 2 because table entries are 2 bytes each.
            LDY #RtoImage ; Point Y to table
            LDY B,Y                    ; Load Y with pointer to proper column of display from the table

            ; NEW
            LDAA 11,X                  ; } Store Zend in V1
            STAA V1                    ; }
            LDAA 5,X                   ; A = Zstart
Vloop       PSHA                       ; PUSH Z
            TAB                        ; B = Z

            ASRA
            ASRA
            ASRA                       ; Database memory offset is in A

            ANDB #%00000111            ; B contains Z value to decode

            LDX #Zdecode
            LDAB B,X                   ; B contains decoded byte

                        ; * X points at proper column *
                        ; * Now use Z to determine which pixel in the column to light up. *

            ORAB A,Y    ; Preserve any pixels already lit up close by.
            STAB A,Y    ;X points to beginning of column denoted by 'r'.
                        ;A offsets the proper quarter of the column.
                        ;B contains the decoded data (1 pixel) to put into that byte.

            PULA                       ; PULL Z
            CMPA V1
            LBEQ LineDone

            INCA                       ; Moving toward Zend, drawing in each pixel.
            BRA Vloop

; * Determine if this is a horizontal line *
TestHoriz   LDY 4,X                    ; Y = Zstart
            CPY 10,X
            BNE FrickinA  ; Branch if NOT horizontal

            ; NEW
            EXG Y,A                    ; A = Z (constant)
            TAB                        ; B = Z

            ASRA
            ASRA
            ASRA                       ; Database memory offset is in *A*

            ANDB #%00000111            ; B contains Z value to decode

            LDY #Zdecode
            LDAB B,Y
            STAB V1                    ; Decoded byte is in *V1*

            ; Find Rstart, increment and loop until it gets to Rend.

            LDAB 1,X                   ; B contains Rstart.
            STAB V2                    ; Initialize *V2* to Rstart (used as counter)
            ASLB                       ; multiply B times 2 because table entries are 2 bytes each.
            LDY #RtoImage ; Point Y to table
            LDY B,Y                    ; Load Y with pointer to proper column of display from the table

            ; * X points at proper column *
            ; * Now use Z to determine which pixel in the column to light up. *

Hloop       LDAB V1
            ORAB A,Y    ; Preserve any pixels already lit up close by.
            STAB A,Y    ;Y points to beginning of column denoted by 'r'.
                        ;A offsets the proper quarter of the column.
                        ;B contains the decoded data (1 pixel) to put into that byte.

            LDAB V2
            CMPB 7,X
            LBEQ LineDone
```

```
                INCB
                STAB V2
                DEY             ; Increment Y by 4
                DEY
                DEY
                DEY
                BRA Hloop

; * Incident line is DIAGONAL across the freaking display *
FrickinA        NOP
                NOP
                NOP

                LDD 0,X
Floop           PSHD            ; PUSH D: Rcurrent
                LDD 6,X
                SUBD 0,X        ; D = Rend - Rstart
                EXG D,Y         ; Y = Rend - Rstart
                LDD 0,SP        ; Refresh Rcurrent from stack, do not pull.
                SUBD 0,X        ; D = Rcurrent - Rstart
                PSHX            ; PUSH X: Coordinate pointer
                EXG Y,X         ; X = Rend - Rstart
                LDY #$200       ; } D = 512 * (Rcurrent - Rstart)
                EMULS           ; }
                IDIVS           ; X = 512 * (Rcurrent - Rstart) / (Rend - Rstart)

                ; X contains the ratio of the current r value being filled out of all the r values that will be
filled.
                ; Note: X is multiplied by 512.

                EXG X,Y         ; Y = 512 * (Rcurrent - Rstart) / (Rend - Rstart)
                LDX 0,SP        ; refresh X: Coordinate Pointer
                LDD 14,X        ; D = Zend - Zstart
                EMULS           ; D = (Zend - Zstart) * (512 * (Rcurrent - Rstart) / (Rend - Rstart))

                ; Divide the 512 back off.
                TFR A,B
                ASRB
                ANDA #%00000001             ; }
                BEQ round                   ; } Rounding properly.
                INCB                        ; }
round           CLRA

                ADDD 4,X        ; D = (Zend - Zstart) * ((Rcurrent - Rstart) / (Rend - Rstart)) + Zstart = Zcurrent
                STD V2          ; Zcurrent is in *V2*

                ; Determine R (column) for display database
                LDD 2,SP                    ; D = Rcurrent (refresh from stack)
                ASLB                        ; multiply B times 2 because table entries are 2 bytes each.
                LDY #RtoImage ; Point Y to table
                LDY B,Y                     ; Load Y with pointer to proper column of display from the table

                ; Determine Z (data for column) for display
                LDD V2                      ; D = Zcurrent
                TBA                         ; A = B = Zcurrent

                ASRA
                ASRA
                ASRA                        ; Database memory offset is in *A*

                ANDB #%00000111             ; B contains Z value to decode

                LDX #Zdecode
                LDAB B,X        ; B contains decoded Z (8-bit)
                ORAB A,Y        ; Preserves any pixels already lit up close by
                STAB A,Y        ; Store decoded Z to display database

                ; Check if done
                PULX                        ; PULL X: Coordinate Pointer
                PULD                        ; PULL D: Rcurrent
                INCB
                CPD 6,X
                BLT Floop
                BRA LineDone

                ; *** CALCULATION: LINE-DISPLAY INTERSECTION IS A POINT ***
                ; * Calculate V1 and V2 *

Intersect       LDD Degree      ; D = T
                SUBD 8,X        ; D = T - Tend
                ADDD #360       ; T - Tend is always negative; making it positive
                LDY #sine
                LDY D,Y         ; Y = $200 * sin (T - Tend + 360)
                LDD 6,X         ; D = Rend
                EMULS           ; D = $200 * sin (T - Tend + 360) * Rend
                STD V1          ; V1 Stored.

                LDD Degree      ; D = T
```

```
              SUBD 2,X      ; D = T - Tstart
              LDY #sine
              LDY D,Y       ; Y = $200 * sin (T - Tstart)
              LDD 0,X       ; D = Rstart
              EMULS         ; D = $200 * sin (T - Tstart) * Rstart
              SUBD V1       ; D = $200 * sin (T - Tstart) * Rstart - $200 * sin (T - Tend + 360) * Rend
              STD V2        ; V2 Stored.

              ; * Calculate Z, store in V1 *

              LDD V1
              LDY 14,X
              EMULS         ; Y:D = V1 * dZ, 32-bit number
              PSHX          ; PUSH: Coordinate Pointer
              LDX V2        ; X = V2
              EDIVS         ; Y = (V1 * dZ) / V2
              LDX 0,SP      ; Refresh Coordinate Pointer into X, leave in stack.
              EXG Y,D       ; D = (V1 * dZ) / V2
              ADDD 10,X     ; D = Zend + ((V1 * dZ) / V2) = Z
              STD V1        ; V1 is now 'Z'.

              ; * Calculate "r". *

              LDD V2
              EXG A,B       ; }
              ASRB          ; }
              CMPB #$00     ; } Divide by 512, faster than using IDIVS
              BGE pos       ; }
              INCB          ; }
pos           SEX B,D       ; Ooo, la la...

              PSHD          ; PUSH: Denominator
              LDD 12,X      ; D = Numerator
              PULX          ; PULL: X = Denominator
              IDIVS         ; X = r

              ; * "r" is now in X. *
              EXG X,B       ; b contains r.
              ASLB          ; multiply B times 2 because table entries are 2 bytes each.
              LDX #RtoImage ; Point X to table
              LDX B,X                    ; Load X with pointer to proper column of display from the table

              ; NEW
              LDAA V1+1                  ; A = Z
              TAB                        ; B = Z

              ASRA
              ASRA
              ASRA                       ; Database memory offset is in A

              ANDB #%00000111            ; B contains Z value to decode

              LDY #Zdecode
              LDAB B,Y                   ; B contains decoded byte

              ; * X points at proper column *
              ; * Now use Z to determine which pixel in the column to light up. *

              ORAB A,X      ; Preserve any pixels already lit up close by.
              STAB A,X      ;X points to beginning of column denoted by 'r'.
                            ;A offsets the proper quarter of the column.
                            ;B contains the decoded data (1 pixel) to put into that byte.

              PULX
LineDone      EXG X,D       ; PULL: D = Coorindate Pointer
              ADDD #16      ; Increment pointer by 16
              EXG D,X       ; X = new coordinate pointer

              PULA          ; PULL: Line Counter, check if there are more lines to draw in the database in this 2
degrees.
              DECA
              LBNE NextLine
Exit          RTI           ; If not, return from interrupt so the database can be refreshed on the display.
;*** END IRQ ISS ***

Clear:        LDY #Image                 ; Clears the Display
              CLRA
              CLRB
clrloop       STD 2,Y+
              CPY #Image+$80
              BLT clrloop
              RTS

Degree        fdb $0166     ; Spin Display Clockwise
              ;fdb $0000    ; Spin Display Counter-Clockwise

Image         fdb $0000, $0000
              fdb $0000, $0000
              fdb $0000, $0000
```

```
            fdb $0000, $0000
            fdb $0000, $0000
            fdb $0000, $0000
            fdb $0000, $0000
            fdb $0000, $0000
            fdb $0000, $0000
            fdb $0000, $0000
            fdb $0000, $0000
            fdb $0000, $0000
            fdb $0000, $0000
            fdb $0000, $0000
            fdb $0000, $0000
            fdb $0000, $0000
            fdb $0000, $0000
            fdb $0000, $0000
            fdb $0000, $0000
            fdb $0000, $0000
            fdb $0000, $0000
            fdb $0000, $0000
            fdb $0000, $0000
            fdb $0000, $0000
            fdb $0000, $0000
            fdb $0000, $0000
            fdb $0000, $0000
            fdb $0000, $0000
            fdb $0000, $0000
            fdb $0000, $0000
            fdb $0000, $0000
            fdb $0000, $0000

            fdb $FFFF, $FFFF            ; (For debug purposes)

sine        FDB 0
            FDB 17
            FDB 35
            FDB 53
            FDB 71
            FDB 88
            FDB 106
            FDB 123
            FDB 141
            FDB 158
            FDB 175
            FDB 191
            FDB 208
            FDB 224
            FDB 240
            FDB 255
            FDB 271
            FDB 286
            FDB 300
            FDB 315
            FDB 329
            FDB 342
            FDB 355
            FDB 368
            FDB 380
            FDB 392
            FDB 403
            FDB 414
            FDB 424
            FDB 434
            FDB 443
            FDB 452
            FDB 460
            FDB 467
            FDB 474
            FDB 481
            FDB 486
            FDB 492
            FDB 496
            FDB 500
            FDB 504
            FDB 507
            FDB 509
            FDB 510
            FDB 511
            FDB 512
            FDB 511
            FDB 510
            FDB 509
            FDB 507
            FDB 504
            FDB 500
            FDB 496
            FDB 492
            FDB 486
            FDB 481
            FDB 474
```

```
FDB 467
FDB 460
FDB 452
FDB 443
FDB 434
FDB 424
FDB 414
FDB 403
FDB 392
FDB 380
FDB 368
FDB 355
FDB 342
FDB 329
FDB 315
FDB 300
FDB 286
FDB 271
FDB 256
FDB 240
FDB 224
FDB 208
FDB 191
FDB 175
FDB 158
FDB 141
FDB 123
FDB 106
FDB 88
FDB 71
FDB 53
FDB 35
FDB 17
FDB 0
FDB -17
FDB -35
FDB -53
FDB -71
FDB -88
FDB -106
FDB -123
FDB -141
FDB -158
FDB -175
FDB -191
FDB -208
FDB -224
FDB -240
FDB -255
FDB -271
FDB -286
FDB -300
FDB -315
FDB -329
FDB -342
FDB -355
FDB -368
FDB -380
FDB -392
FDB -403
FDB -414
FDB -424
FDB -434
FDB -443
FDB -452
FDB -460
FDB -467
FDB -474
FDB -481
FDB -486
FDB -492
FDB -496
FDB -500
FDB -504
FDB -507
FDB -509
FDB -510
FDB -511
FDB -512
FDB -511
FDB -510
FDB -509
FDB -507
FDB -504
FDB -500
FDB -496
FDB -492
FDB -486
FDB -481
```

```
                FDB -474
                FDB -467
                FDB -460
                FDB -452
                FDB -443
                FDB -434
                FDB -424
                FDB -414
                FDB -403
                FDB -392
                FDB -380
                FDB -368
                FDB -355
                FDB -342
                FDB -329
                FDB -315
                FDB -300
                FDB -286
                FDB -271
                FDB -256
                FDB -240
                FDB -224
                FDB -208
                FDB -191
                FDB -175
                FDB -158
                FDB -141
                FDB -123
                FDB -106
                FDB -88
                FDB -71
                FDB -53
                FDB -35
                FDB -17
                FDB 0

RtoImage        FDB Image+128
                FDB Image+124
                FDB Image+120
                FDB Image+116
                FDB Image+112
                FDB Image+108
                FDB Image+104
                FDB Image+100
                FDB Image+96
                FDB Image+92
                FDB Image+88
                FDB Image+84
                FDB Image+80
                FDB Image+76
                FDB Image+72
                FDB Image+68
                FDB Image+64
                FDB Image+60
                FDB Image+56
                FDB Image+52
                FDB Image+48
                FDB Image+44
                FDB Image+40
                FDB Image+36
                FDB Image+32
                FDB Image+28
                FDB Image+24
                FDB Image+20
                FDB Image+16
                FDB Image+12
                FDB Image+8
                FDB Image+4
                FDB Image+0

Zdecode         fcb %00000001
                fcb %00000010
                fcb %00000100
                fcb %00001000
                fcb %00010000
                fcb %00100000
                fcb %01000000
                fcb %10000000

V1              rmb 2
V2              rmb 2

AnimCnt         fcb 15
AnimDirec       fcb 0

PointCnt        fcb 29 ;28 good, 29

Points          fdb 30       ; Base
                fdb 0
```

```
fdb 0
fdb 30
fdb 90
fdb 0
rmb 4

fdb 30
fdb 90
fdb 0
fdb 30
fdb 180
fdb 0
rmb 4

fdb 30
fdb 180
fdb 0
fdb 30
fdb 270
fdb 0
rmb 4

fdb 30
fdb 270
fdb 0
fdb 30
fdb 360
fdb 0
rmb 4

fdb 0         ; diagonal intersecting lines
fdb 0
fdb 30
fdb 30
fdb 0
fdb 0
rmb 4

fdb 0
fdb 90
fdb 30
fdb 30
fdb 90
fdb 0
rmb 4

fdb 0
fdb 180
fdb 30
fdb 30
fdb 180
fdb 0
rmb 4

fdb 0
fdb 270
fdb 30
fdb 30
fdb 270
fdb 0
rmb 4

fdb 0         ; Bottom Face Diagonals
fdb 0
fdb 0
fdb 30
fdb 0
fdb 0
rmb 4

fdb 0
fdb 90
fdb 0
fdb 30
fdb 90
fdb 0
rmb 4

fdb 0
fdb 180
fdb 0
fdb 30
fdb 180
fdb 0
rmb 4

fdb 0
fdb 270
fdb 0
```

```
            fdb 30
            fdb 270
            fdb 0
            rmb 4

            fdb 0          ; triangle face vertical diagonals
            fdb 44
            fdb 30
            fdb 21
            fdb 44
            fdb 0
            rmb 4

            fdb 0
            fdb 134
            fdb 30
            fdb 21
            fdb 134
            fdb 0
            rmb 4

            fdb 0
            fdb 224
            fdb 30
            fdb 21
            fdb 224
            fdb 0
            rmb 4

            fdb 0
            fdb 314
            fdb 30
            fdb 21
            fdb 314
            fdb 0
            rmb 4

            fdb 15         ; triangle face horizontal diagonals
            fdb 0
            fdb 15
            fdb 15
            fdb 90
            fdb 15
            rmb 4

            fdb 15
            fdb 90
            fdb 15
            fdb 15
            fdb 180
            fdb 15
            rmb 4

            fdb 15
            fdb 180
            fdb 15
            fdb 15
            fdb 270
            fdb 15
            rmb 4

            fdb 15
            fdb 270
            fdb 15
            fdb 15
            fdb 360
            fdb 15
            rmb 4

; *** ANIMATED TRIANGULAR PRISM ***

            fdb 21         ; Bottom
            fdb 180
Animated    fdb 0
            fdb 30
            fdb 224
            fdb 0
            rmb 4

            fdb 21
            fdb 180
            fdb 0
            fdb 21
            fdb 270
            fdb 8
            rmb 4

            fdb 30
            fdb 224
```

```
              fdb 0
              fdb 21
              fdb 270
              fdb 8
              rmb 4

              fdb 21        ; Top
              fdb 180
              fdb 8
              fdb 30
              fdb 224
              fdb 8
              rmb 4

              fdb 21
              fdb 180
              fdb 8
              fdb 21
              fdb 270
              fdb 16
              rmb 4

              fdb 30
              fdb 224
              fdb 8
              fdb 21
              fdb 270
              fdb 16
              rmb 4

              fdb 21        ; Edges
              fdb 180
              fdb 0
              fdb 21
              fdb 180
              fdb 8
              rmb 4

              fdb 30
              fdb 224
              fdb 0
              fdb 30
              fdb 224
              fdb 8
              rmb 4

              fdb 21
              fdb 270
              fdb 8
              fdb 21
              fdb 270
              fdb 16
              rmb 4

*** Main Program Zone ***

              org $2000

              LDS #$1FFF

              BSET DDRA,$FF
              BSET DDRT,$FF
              BSET DDRP,$FF
              BSET DDRM,$FF
              BSET DDRH,$FF

              BSET PortA,$FF
              CLR PortT
              CLR PortP
              CLR PortM
              CLR PortH

              BSET IntCR,#$80            ; Set !IRQ to be triggered on falling edge.

;*** Precalculate ***

              ; * Numerator of 'r' *
              LDX #Points   ; Point X at first line segment
              LDAA PointCnt
CalcNext      PSHA
              LDD 8,X
              SUBD 2,X      ; D = Tend - Tstart
              LDY #sine
              LDD D,Y       ; D = $200 * sin (Tend - Tstart)
              LDY 6,X       ; Y = Rend
              EMULS         ; D = $200 * sin (Tend - Tstart) * Rend
              LDY 0,X       ; Y = Rstart
              EMULS         ; Y:D = $200 * sin (Tend - Tstart) * Rend * Rstart
```

```
              PSHX
              LDX #$200
              EDIVS         ; Y = sin (Tend - Tstart) * Rend * Rstart
              PULX

              STY 12,X      ; Precalculated Numerator Stored.

              ; * Zend - Zstart *
              LDD 10,X      ; D = Zend
              SUBD 4,X      ; D = Zend - Zstart
              STD 14,X      ; Precalculated Zend - Zstart Stored.

              ; * Update Pointer *
              EXG X,D
              ADDD #16
              EXG D,X       ; X points at next line segment

              PULA
              DECA                        ; } Check to see if all line segments are done.
              BNE CalcNext  ; }

;*** End Precalculate ***

              LDD #IRQ_ISS  ; } Connect IRQVector to IRQ_ISS subroutine
              STD IRQVector ; }

              ;LDD #XIRQISS ; } Reroute abort button to clear display before entering debugger
              ;STD XIRQVector             ; }

              CLI                         ; Enable Interrupts

ScanHome      CLR PortA
              LDX #Image
NextCol       INC PortA

              LDD 2,X+
              STAA PortT
              STAB PortP

              LDD 2,X+
              STAA PortM
              STAB PortH

              CLR PortT
              CLR PortP
              CLR PortM
              CLR PortH

              CPX #Image+128
              BEQ ScanHome
              BRA NextCol

; *** END MAIN PROGRAM ***

; *** Set Abort button to safely clear display before stopping, avoid burn.
XIRQISS:      CLR PortT
              CLR PortP
              CLR PortM
              CLR PortH

              SWI           ; Enter debugger
              RTI           ; Return from debugger
```