westgate-works.com

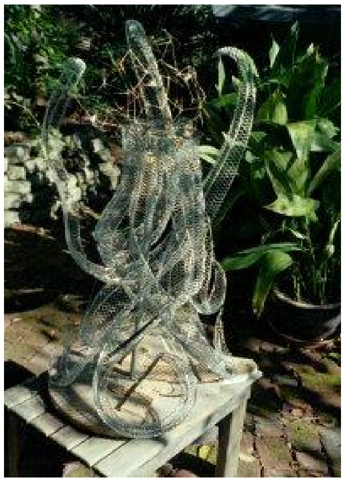
# Fractal LED tutorial

This tutorial covers some of the fabrication of the Dr. Livingston sculpture - including how the fractal head was made, but particularly the replacement of the low wattage light bulb in the fractal head with LED lights.

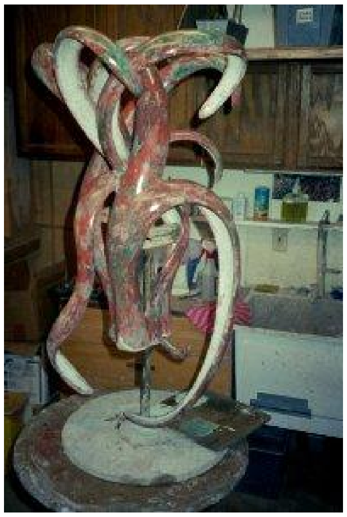
The Dr. Livingstone sculpture body was fabricated using ferrocement which is covered in detail in other tutorials that can be viewed here:
http://www.westgate-works.com/efiles/etutorial.html

The fractal head for this sculpture was made of thin slices of natural stone originally designed for making 35mm slides viewable in a projector. The sculpture stones were salvaged from the scraps left over . These slides and stones were inherited by my father and then myself from the person that actually made the natural stone slides – can't remember his name. You can view some of the stone slides here:
http://www.westgate-works.com/efiles/egodstone.html

This shows the expanded metal armature that was used for applying cement mortar and colored cement to make the sculpture.

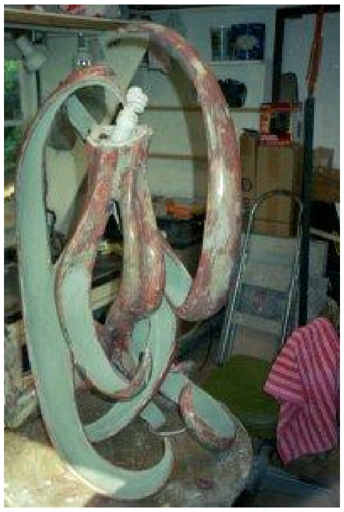


This shows the finished sculpture body mounted upside-down on a work table. The final coats of colored cement have been applied.

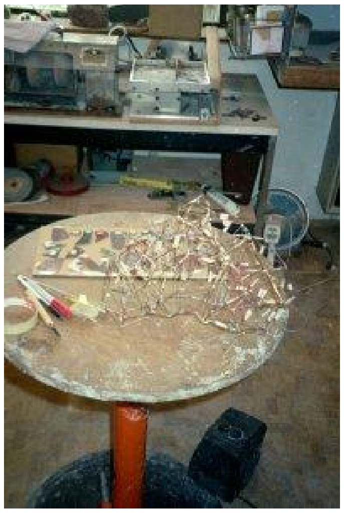


This shows the completed sculpture in it's proper orientation with the low wattage bulb  installed to fit inside the head.

The on/off switch is part of the bulb holder and the chain hangs  through a hole in the body of the sculpture - accessible but out of sight. The power cord for the light feeds through one of the legs.



The fractal head was made of thin slices of natural  stone held in a copper wire frame. The copper wire frame took a few tries before it worked because of soldering at the joints. The solution was to use solder that melted at four different temperatures. By using the hottest solder first, the later soldering would not melt the previous work.

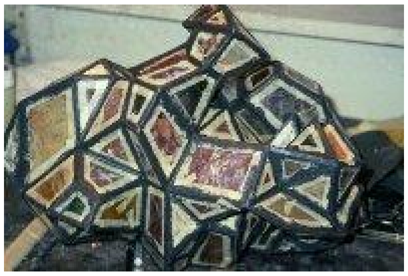


The shape of the head was somewhat dependent on the size and shape of the stones that were available. This often required finding a stone that would fill a spot and then building the frame to match, or vise-versa. There was a lot of lapidary cutting and grinding involved to get things to come together.

Both the shaped stone and matching opening were labeled to keep things organized, as can be seen from the board with the stones taped in numerical order.

This shows the process of gluing the stones in the frame. Originally bronze colored latex caulk was used, but somewhere into the process the stones began falling out of the frame. The culprit was that the latex caulk was reacting with the bare copper and would peel off. All of the stones had to be removed - and kept organized - while painting the entire frame with an oil base paint. As an additional precaution, an oil based caulk was used.

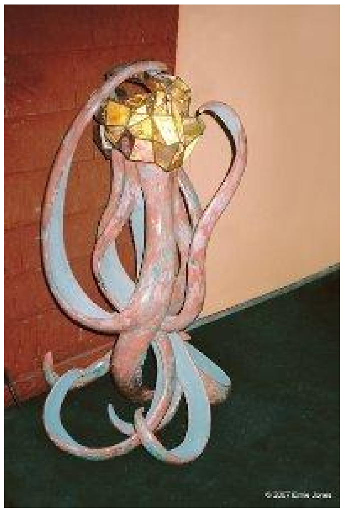
After all of the stones were in place, additional caulking was needed to fill small voids and cracks.  The edges of each stone were masked, and caulk was used to finish the joints and make it as waterproof as possible.



After the caulk dried, the masking tape was remove to reveal nice clean lines, as can be seen at the bottom of this image.

The caulking has survived over 6 years, although it is not waterproof as expected, but is tough and has held the stones in place very well.

This is the finished sculpture in it's original form with the fluorescent light installed in the head.



## Requirements for LED implementation

The low wattage fluorescent light was never a satisfactory solution, and having a light that turned on and off gradually was always the desired implementation. Modifying the sculpture to use  Light Emitting Diodes (LEDs) was not an option, for several reasons, until very recently.

An electronics engineer friend suggested looking into the  Raspberry Pi micro controller as a means for controlling LED lights. Doing research on the web led to learning about the similar Arduino micro controller, which could be powered on and off without problems, unlike the much more versatile and powerful Raspberry Pi micro controller.

Several possible Arduino solutions were identified through researching the web. Most of these solutions required building electronic circuits with chips, resistors and whatnot, and increased the complexity of a solution many times over. Several ready-made solutions were discovered . The simplest was obsolete and no longer available.  The Rainbowduino and accompanying LED shield (a printed circuit board (PCB) that fits on the micro controller PCB) from http://www.seeedstudio.com/ was eventually chosen because it is capable of controlling 192 individual LED's, which is more than are required in the fractal head. However,  Seeed Studio does not provide current, or complete documentation for this product and it was particularly designed for Red  Green Blue (RGB) LED cubes, but it was still the best solution found for this sculpture.  To learn how to use it required a lot of web research, and early on it was decided to not be in a hurry because this technology presented a steep learning curve.

Light Emitting Diodes (LED's) offer options not available  with other types of lighting.  The original plan was to have the light gradually come on when a person approached the sculpture and gradually dim afterwards, but this is difficult to do with fluorescent or incandescent lights, other than manually.

LED's can be dimmed with a pulse width controller which turns them on and off very quickly, making it possible to dim the lights or actually blink the lights or make them twinkle,

It was decided to control the LED's by sensing motion near the sculpture. When motion is sensed, the LED's will randomly start from dim and get brighter the longer motion is present until (if enough time) they are all turned on to full brightness. During this ramp up, LED's are turned on, but not off, so those at a lesser brightness will stay that way until (or if) they are randomly re-set. When motion stops, the reverse is done until all LED's are turned off . It seemed that it would be much more interesting if the lights gradually blinked up and actually twinkled while a person was near and then gradually blinked down to off when no more motion was detected.

Several methods for sensing motion with the Arduino were investigated. The ones that would work with the sculpture were Photocells (CDs) and Passive Infra Red (PIR) Sensors. Other sensors were too bulky and could not be incorporated into the sculpture or had other disadvantages.

Photocells only work in daylight, as they change resistance depending on the  Infra Red light they are receiving. PIR sensors only work in low light or darkness because they sense changes in Infra Red light levels between two sensors and bright daylight saturates the sensors so they do not work properly.

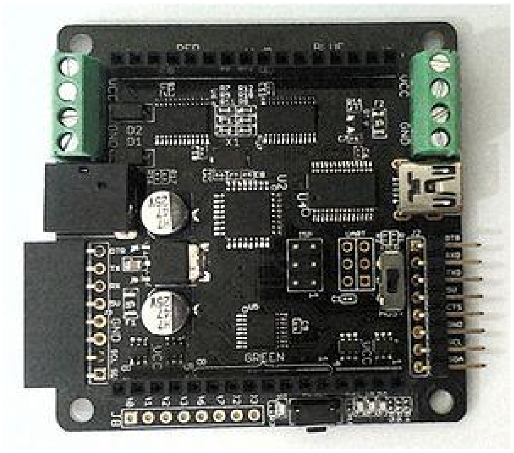
Installing the photocell sensors was simple because they are  very small and almost indestructible and could be unobtrusively mounted on the fractal head surface. The PIR sensors presented a problem because the small sensor element is mounted on a printed PCB. The sensor element is about the same size as the photocell sensor. An attempt was made to separate the sensor unit from the PCB and connect it with very small wires so that the sensors could be mounted the same as the photocell sensors. After much testing it was determined that they would only work dependably as manufactured. The eventual solution was to support 3 PIR senor PCBs under the head of the sculpture and have them peek out between the legs so that they would not be too visible, but still have some capacity to sense motion.

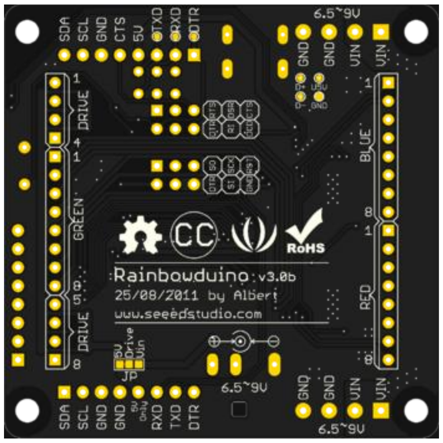
## Hardware technical details

The  Rainbowduino board (http://www.seeedstudio.com/depot/Rainbowduino-LED-driver-platform-Atmega-328-p-371.html) has a built in USB port for power and programming, and a 5 mm 5 volt power socket for stand-alone use and a switch to choose between the two power modes. The Atmega 328 processor includes 32 kilobytes of programmable flash memory, and 2 kilobytes of EPROM memory. The  Rainbowduino board has drivers for 192 LEDs that can be individually set at 16 different levels of  brightness.

The board has 8 labeled connectors without a header (shown below), but there is no documentation to be found other than through Lady Google. A lot of searching turned up  the information that these pins can be used for external control and consist of Arduino pin numbers A0 through A3, A6, A7, D2 and D3. Pins A0 through A3 are standard analog pins that can also be used as input/output digital pins. Pins A6 and A7 are floating analog pins that can only be used for analog input - not output, because they are connected directly to the internal Analog Digital Converter. These two pins are only available on surface mounted versions of the Atmega 328 processor. Pins D2 and D3 are digital pins normally used as interrupts. They cannot be pulled HIGH, but can be pulled LOW. Pin D3 is a Pulse Width Modulation (PWM) pin. An 8 pin female angle header was installed and the pins are used as follows:

A0 - CDs sensor with 10K Ohm resistor to a ground pin on the Rainbowduino PCB
A1 - + 5V for CDs sensor
A2 - + 5V for PIR sensors
A3 - PIR 1 sensor
A6 - PIR 2 sensor
A7 - PIR 3 sensor
D2 - Gnd for PIR sensors
D3 - not used

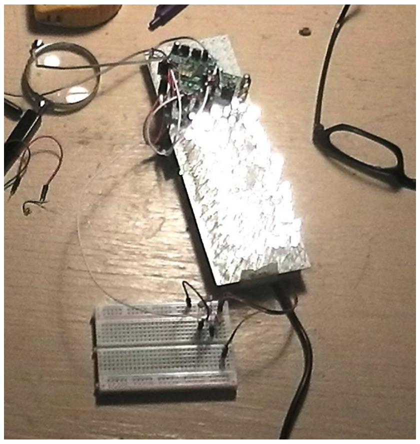

8 connectors at bottom of this top view



8 connectors at left of this bottom view

The LED shield comes with headers that need to be soldered in place, and a full compliment of LED's. (http://www.seeedstudio.com/wiki/Rainbowduino_Extension_Board_v0.9b)

The LED connections are nicely identified individually on the board, but there is no indication as to which side of the board the headers are to be installed, nor is there any indication of the proper direction to install the shield on the Rainbowduino board, but it only works one way. Looking through the PCB with a bright light, the copper traces can be seen and comparing it to the labels on the Rainbowduino board provides the necessary orientation information. A marker pen makes it easy to keep oriented after that.


For testing purposes 120 LED's were installed on the shield by folding and crimping the ends and inserting them in the holes so they could be removed later.

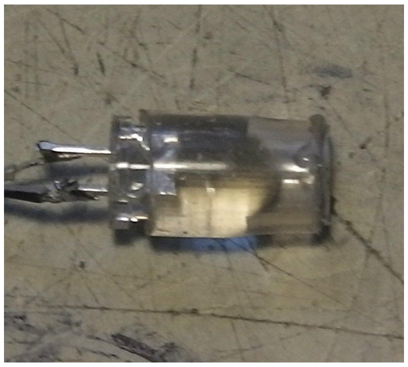
This image shows the LEDs all on and the 3 PIR PCBs are mounted on the shield with their sensors connected by wires – this was later abandoned as not workable.

The breadboard was used here to test one Photocell. The final setup uses 6 photocells wired in series to attain more accuracy. Small holes were drilled around the head through the stone panes for the photocell wires.

Originally 3mm bright warm white LEDs were purchased, but did not seem bright enough, and were replaced with 5mm LEDs.

Fiddling around with an illumined LED inside the head showed that much of the light was being absorbed by the dark area in front of the LEDs. To solve this, each LED was inserted in a short piece of vinyl tubing as a standoff and a cupped reflective sequin was glued to the end of the tubing to reflect the light, which worked better.

All 120 LEDs were assembled like this.



A couple of the stone panes had to be removed for access, as shown here.

The LEDs were glued inside the head at each apex or joining of several stone panes. A clear silicone caulk was used and the wires were taped to hold the LEDs in place until the caulk dried, which was a slow process.  Only a few LEDs could be put in place each day.

The wires were relocated inside before replacing the stone panes.

This soldering station setup had holes drilled for holding each LED assembly for soldering the lead wires to it. A 3.5 volt DC power supply with a resistor was used  to light each LED during soldering and again when the LED was glued in the head to ensure that no mistakes in wiring or placement happened.

The two upright wood pieces were designed for holding the shield during soldering, but proved to be useless.

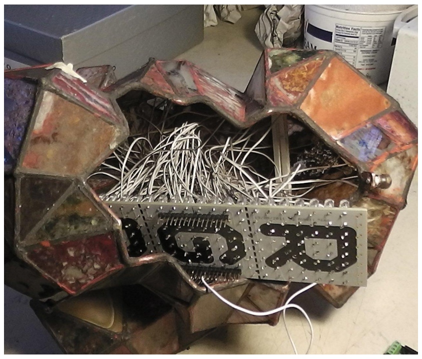
This shows that all of the LEDs have been glued in place and the wires cut to length for soldering to the shield.

The wires are salvaged from an 80 pair hard drive flat ribbon cable and are very small and fragile. To identify polarity for each pair, the cable was rolled up and a black marker pen was used to color one edge. Then two wires were peeled off to make a pair.

The two white wires shown are the connectors to the 6 photocell sensors and are soldered to a 2-pin male plug.

This shows all of the wires soldered to the LED shield. The end of each wire had to be stripped and pre-soldered to get the strands to go into the holes on the shield. Once inserted, they were soldered from the other side of the shield.

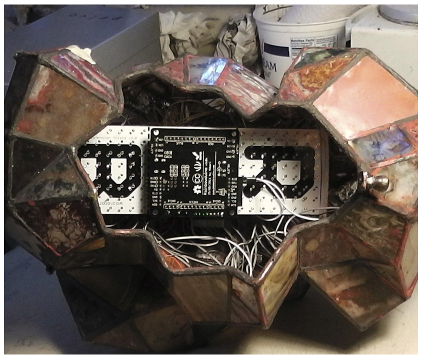
All remaining positions have LEDs soldered on the shield to increase the amount of light, except the end rows, which needed to be clear for mounting the shield in the head. One of the PVC mounting strips with a groove cut in it for holding the shield is visible. These PVC strips were glued in place.



This shows the shield with the Rainbowduino installed and in place in the head.

The two white wires for the photocells are connected to pins A0 & A1 of the 8 pin angle header.

The PIR sensor connector fits the remaining pins on the header.



This shows the upper portion of the sculpture body that the fractal head fits over. A metal plate is fitted into the bottom of the cavity and the 3 PIR sensors are supported underneath. The PIR wiring feeds through to the connector. The 9 volt power cord uses the existing tube that feeds from the bottom of the sculpture.

This shows the fractal head on the sculpture body with the connectors in place, ready to be fitted over the sculpture body.

These three images show the fractal head in operation – morning, noon and night.

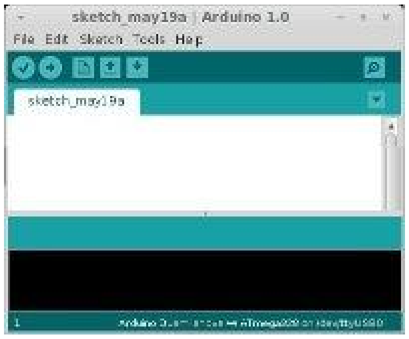
**Programming the Rainbowduino**
using LINUX Mint 13

Details of using Linux Mint 13 are not included because it is beyond the scope of this tutorial. Other operating systems will have different requirements but similar processes.

The Rainbowduino has an Arduino boot-loader already installed that makes it easy to program using the Integrated Development Environment (IDE) program and a USB connection.

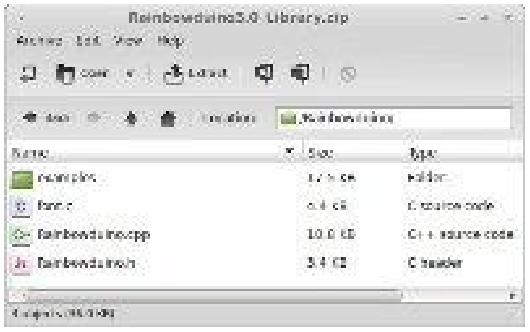
The Arduino IDE for Linux Mint 13 is included in the distribution's program library and can be installed from the Software Manger or the Package Manager or by keying in "Arduino IDE" In the Menu search box and installing from there.

Opening the Arduino IDE displays this window.


sketch_may19a | Arduino 1.0
File Edit Sketch Tools Help
sketch_may19a
1
Arduino [illegible] ATmega328 on /dev/ttyUSB0

Programs in the IDE are called "Sketches" and are stored (in Linux Mint 13) in a Sketchbook folder located at userhome/sketchbook.

The Rainbowduino programming library needs to be downloaded from this web page [http://www.seeedstudio.com/wiki/Rainbowduino_v3.0](http://www.seeedstudio.com/wiki/Rainbowduino_v3.0) .



| Name | Size | Type |
|---|---|---|
| examples | [illegible] | folder |
| font.c | [illegible] | C source code |
| Rainbowduino.cpp | 10.6 kB | C++ source code |
| Rainbowduino.h | 3.4 kB | C header |

Under the "RESOURCES" heading at the bottom, select "Rainbowduino3.0 Library for Arduino 023" to download the PDF file.

The downloaded "Rainbowduino3.0_Library.zip" contains these files and folders when unzipped.

The Rainbowduino programming library does not include a function for lighting all of the LEDs and is required for this project. Therefore it needed to be added BEFORE importing the Rainbowduino library into the Arduino IDE libraries, requiring that the changes had to be made in the zipped library.

To ensure that the modified code would compile, Rainbowduino.cpp was opened in a text editor and copied and pasted into the IDE to make the following changes;

In the IDE, the following  Rainbowduino.cpp program code was copied :

```
//blank all pixels

void Rainbowduino::blankDisplay(void)
{
  for(unsigned char x=0;x<=7;x++)
    {
     for(unsigned char y=0;y<=7;y++)
      {
       frameBuffer[0][x][y] = 0x00;
       frameBuffer[1][x][y] = 0x00;
       frameBuffer[2][x][y] = 0x00;
      }
    }
}
```

and then pasted just below where it was copied from (so that the copy could be changed).
In the pasted copy of the code;

“//blank all pixels ” was changed to “//turn on all pixels ”.
“void Rainbowduino::blankDisplay(void)” was changed to “void Rainbowduino::allonDisplay(void)”.
All three instances of “0x00”  were changed to “0xFF”.

The code was compiled to be sure it was O.K.
All of the modified code from the IDE was copied back into the text editor (replacing the original code). The modified  Rainbowduino.cpp was saved with the text editor (writing over the existing file in the library). These changes added the RB.allonDisplay() function to the Rainbowduino.cpp code.

All of the code in the IDE was cleared.

Rainbowduino.h was opened in the text editor and all of the code was copied and pasted into the IDE for making the following changes;

In the IDE program, that had the Rainbowduino.h program code;
The code “void blankDisplay(void);”  was copied and then pasted  just below where it was copied (so that it could be changed).
“void blankDisplay(void)” was changed to “void allonDisplay(void);”
The code was compiled to be sure it was O.K.
All of the modified code from the IDE was copied back into the text editor (to replace the original code). The modified  Rainbowduino.h was saved with the text editor (writing over the existing file in the library). These changes added the RB.allonDisplay() function to Rainbowduino.h code.

To add the modified “Rainbowduino3.0_Library.zip” to the IDE library, this web page was used as a guide: http://arduino.cc/en/Guide/Libraries The Arduino libraries in Linux Mint 13 are located in usr/share/arduino/libraries.

Before uploading a program (sketch) to the Rainbowduino board via the USB connection, the IDE required selecting an Arduino board. This was done by using the menu item Tools > Board > and then selecting Duemilanova W/ ATmega328.

The process of writing and testing the code (Sketch) for this fractal head was several months long with many changes.

This is the complete working code as uploaded into the Rainbowduino flash memory and is published here in the hope that it may be of some use to others.

```
//
// PIR sensor code from  http: //playground.arduino.cc/Code/PIRsense
// CDs sensor code from https: //learn.adafruit.com/photocells
// LED code Rainbowduino v3.0 Library examples Moodlamp - Rb.setPixel(XYZ)
// 3 PIR sensors are read sequentially.
// PIR Sensor PIR-HC-SR501 has an internal process time of 1 second.
// PIR sensor sensitivity increases with less light and lower temperature
// CDs sensor sensitivity increases with more light
// 6 CDs sensors connected in series between +5 Volts and Pin A0.
// Pin A0 connected with 10K ohm resistor to ground.
// CDs  sensors calibrate to about: Dark~ 0 Full sunlight ~ 1000.
// 10 LED's are processed together.
// LED lights gradually randomly blink up - to all on, taking 3 seconds.
// LED lights randomly twinkle while motion is still being detected.
// LED lights gradually randomly blink down at end of motion - to all off.
// Rainbowduino shield has 120 LED's wired to fractal head vertices.
// Remaining on-board LED positions are populated for more light
//
// Rainbowduino pins used:
// A0 - CDs sensor with 10K ohm resistor to a ground pin on the PCB
// A1 - + 5V for CDs sensor -  CDs Vcc
// A2 - + 5V for PIR sensors - PIR Vcc
// A3 - PIR 1 sensor
// A6 - PIR 2 sensor
// A7 - PIR 3 sensor
// D2 - Gnd for PIR sensors
// D3 - not used

#include <Rainbowduino.h> // Include the modified Rainbowduino library

//  =================   Program controls =================
// CDs sensors are made active above early evening light level and
// PIR sensors are made active 10 points below this same changeover level
// to reduce chance of having both CDs and PIR running at same time
 int CDslevel = 400;
 int PIRlevel = 390;

// Rapid light change across changeover value allows runaway active
// process between PIR and CDs and is resolved by resetting ambiant
// light level to longer than a normal ON time
 int reset = 50;
 int loopCount = 1;

// Trigger level for CDs motion sensing needs to be above voltage jitter level
// 5 allows sensing at about 4 feet in daylight, less as it gets darker.
// In bright light voltage jitter increases so this value is increased to 8.
int CDsTrigger = 5;

// The amount of milliseconds the sensors have to be low
// before we assume all motion has stopped.
// During this time blinkUp() uses about 3 seconds - so this makes 6 seconds total
 long unsigned int pause = 3000;

 int AmbientCDs;    // Ambient CDs reading
 int CDsVal;        // Current CDs reading
 int pir1;          // PIR 1 value
 int pir2;          // PIR 2 value
 int pir3;          // PIR 3 value
 int pirPin;        // PIR combined value
 int j;             // Microseconds delay between each LED brightness change
 int w;             // Count 0-9 for LED x & y array
 int x[10], y[10];  // Two sets of 10 random numbers
 long Rseed = 1;    // Counter for randomseed
 int z;             // LED brightness

//the time when the sensor outputs a low impulse
 long unsigned int PIRlowIn;
 long unsigned int CDslowIn;
 boolean PIRlockLow = true;  // Flip-Flop for PIR
 boolean PIRtakeLowTime;     // Delay time for PIR
 boolean CDslockLow = true;  // Flip-Flop for CDs
 boolean CDstakeLowTime;     // Delay time for CDs

void setup()
{
  Rb.init();              //initialize Rainbowduino driver
//  Serial.begin(9600);   // Debug

  pinMode(A1, OUTPUT);    // CDs Vcc
  digitalWrite(A1, HIGH); // CDs Vcc
  pinMode(A2, OUTPUT);    // PIR Vcc
  digitalWrite(A2, HIGH); // PIR Vcc
  pinMode(2, OUTPUT);     // PIR Gnd
  digitalWrite(2, LOW);   // PIR Gnd

  AmbientCDs = (analogRead(A0)); // Start ambient CDs light level
}

void loop()  // MAIN loop
```

```
{
  CDsVal = (analogRead(A0));  //  Read CDs sensors

  if (CDsVal > 600) // Adjust trigger for bright light
   { CDsTrigger=8; }
   else
   { CDsTrigger=5; }

/*  Debug
        Serial.print(AmbientCDs);
        Serial.print(" = ");
        Serial.print(CDsVal);
        Serial.print(" loop = ");
        Serial.println(loopCount);
*/

  loopCount++;
  if (loopCount==reset)         // Reset ambient light level to avoid runaway
   {
    AmbientCDs=CDsVal;
    loopCount=1;
   }

  if (AmbientCDs>CDslevel)
   {
     CDsSense();
   }

  if (AmbientCDs<PIRlevel)
   {
       PIRsense();
   }
}  // =========== END OF MAIN LOOP ===============

int CDsSense() // CDs sensor function
 {
  /*  Debug
         Serial.print(AmbientCDs<CDsVal-10);
         Serial.print(" = Ambient-10  compare CDsVal+10 = ");
         Serial.print(AmbientCDs>CDsVal+10);
         Serial.print(" loop = ");
         Serial.println(loopCount);
  */
   if ( AmbientCDs<CDsVal-CDsTrigger || AmbientCDs>CDsVal+CDsTrigger ) // True/False 1 or 0
   {
    if(CDslockLow)    //makes sure we wait for a transition to LOW before any further output is made
     {
     CDslockLow = false;
     blinkUp();          // Motion detected - turn it on
     Rb.allonDisplay();
     }
    CDstakeLowTime = true;

    }

  if(!CDslockLow) { j=50; Twinkle();}

  CDsVal = (analogRead(A0));  //  Re-read CDs sensors

  if ( !(AmbientCDs<CDsVal-CDsTrigger || AmbientCDs>CDsVal+CDsTrigger) )
    {
     if(CDstakeLowTime){
     CDslowIn = millis();  //save the time of the transition from high to LOW
     CDstakeLowTime = false;  //make sure this is only done at the start of a LOW phase
    }
                  // If the sensor is low for more than the given pause,
                  // we assume that no more motion is going to happen
     if(!CDslockLow && millis() - CDslowIn > pause)
        {
                  // Makes sure this block of code is only executed again after
                  // a new motion sequence has been detected
         CDslockLow = true;
                  //  Motion ended so blink down
         blinkDown();
         Rb.blankDisplay();
       AmbientCDs=(analogRead(A0));  // End of motion so reset ambient light level

  /*  Debug
        Serial.print(AmbientCDs);
        Serial.print(" - Ambient reset - ");
        Serial.println(CDsVal);
        Serial.print("------ CDs -  motion ended at ");
        Serial.print((millis() - pause)/1000);
        Serial.println(" sec");
 */

         }
     }
}  //==============  end of CDs Sensor function =============

int PIRsense()   // PIR sensor function

{
                    //  Read 3 sensors and add together
    pir1 = (analogRead(A3));
    pir2 = (analogRead(A6));
    pir3 = (analogRead(A7));
    pirPin = (pir1 + pir2 + pir3);

/*   Debug
         Serial.print(" PIR1= ");
         Serial.print(pir1);
         Serial.print(" PIR2= ");
         Serial.print(pir2);
         Serial.print(" PIR3= ");
         Serial.println(pir3);
 */
   if (pirPin>10)       //  Upped from 0 to 10 because of noise
    {
     if(PIRlockLow)      //makes sure we wait for a transition to LOW before any further output is
made
      {
       PIRlockLow = false;
                    // Motion detected so blink up
/*  Debug
       Serial.print("------ PIR  motion detected at ");
       Serial.print((millis() - pause)/1000);
       Serial.println(" sec");
 */
       blinkUp();
       Rb.allonDisplay();
      }
       PIRtakeLowTime = true;
    }

    if(!PIRlockLow) { j=50; Twinkle();}
                      // Read 3 sensors and add together
    pir1 = (analogRead(A3));
    pir2 = (analogRead(A6));
    pir3 = (analogRead(A7));
    pirPin = (pir1 + pir2 + pir3);

   if(pirPin==0)
     {
     if(PIRtakeLowTime){
      PIRlowIn = millis();     //save the time of the transition from high to LOW
      PIRtakeLowTime = false;  //make sure this is only done at the start of a LOW phase
      }
                      // If the sensor is low for more than the given pause,
                      // we assume that no more motion is going to happen
     if(!PIRlockLow && millis() - PIRlowIn > pause)
        {
                      // Makes sure this block of code is only executed again after
                      // a new motion sequence has been detected
         PIRlockLow = true;
                      //  Motion ended so blink down
         blinkDown();
         Rb.blankDisplay();
         AmbientCDs=CDsVal;  // Reset ambiant light level for CDs sensing

/*  Debug
       Serial.print(AmbientCDs);
       Serial.print(" - Ambient reset - ");
```

```
        Serial.println(CDsVal);
        Serial.print("------ PIR -----  motion ended at ");
        Serial.print((millis() - pause)/1000);
        Serial.println(" sec");
  */
         }
       }
  } //============  end of PIR Sensor function ==========

int blinkUp()        // Reduce delay time to increase speed for Blink up
{
 j=95;
 while (j>0)
  {
    j=j-5;
    LEDsUp();
  }
}  // ============  End Blink UP function  ==============

int Twinkle()
{
  rand();               // Save random numbers
  for(z=100; z<255; z++) // Up Bright - z is LED brightness (0>255)
    {
     for (w=0; w<10; w++)// w count 0-9 for x & y arrays
      {
       Rb.setPixelXY(x[w],y[w],z,z,z);
       if (j==0){ return 0; }  // Eliminates delay = 0 which occurs HERE
       delayMicroseconds(j);
      }
    }

  for(z=255; z>1000; z--) // DOWN dim - z is LED dimness (255>0)
    {
     for (w=0; w<10; w++) // w count 0-9 for x & y arrays
       {
        Rb.setPixelXY(x[w],y[w],z,z,z);
        if (j==0){ return 0; }  // Eliminates delay = 0
        delayMicroseconds(j);
       }
    }
}  // =============  End Twinkle function  =============

int blinkDown()
{
 j=0;
 while (j<85)
  {
    j=j+5;           // Increase delay time to reduce speed for Blink down
    LEDsDown();
  }
```

```
}  // ============= End Blink Down function    ===========

int rand() // Save random numbers for both x and y
  {

    if (Rseed == 4294967294)
       {Rseed =1;}

    Rseed++;
    randomSeed(Rseed);
    for (w=0; w<10; w++)
     {
      x[w]=random(8);
      y[w]=random(8);
     }
  }  //  ===========  end Rand function ============

int LEDsUp()
{
  rand();              // Get random numbers
  for(z=0; z<255; z++)   // Up Bright - z is LED brightness (0 - 255)
    {
    for (w=0; w<10; w++) // w count 0-9 for x & y arrays
     {
      Rb.setPixelXY(x[w],y[w],z,z,z);
      if (j==0){ return 0; }  // Eliminates delay = 0
      delayMicroseconds(j);
     }
    }
}  //  ============== end LEDsUp function  ==========
int LEDsDown()
{
  rand();              // Get random numbers
  for(z=255; z>0; z--)   // DOWN dim - z is LED dimness (255 - 0)
    {
     for (w=0; w<10; w++) // w count 0-9 for x & y arrays
      {
       Rb.setPixelXY(x[w],y[w],z,z,z);
       if (j==0){ return 0; } // Eliminates delay = 0
       delayMicroseconds(j);
      }
    }
}  //  ============== end LEDsDown function  ==========
int blinkDown()
{
 j=0;
 while (j<85)
  {
    j=j+5;            // Increase delay time to reduce speed for Blink down
    LEDsDown();
  }
```

```
}  // ============= End Blink Down function    ===========

int rand() // Save random numbers for both x and y
  {

     if (Rseed == 4294967294)
        {Rseed =1;}

     Rseed++;
     randomSeed(Rseed);
     for (w=0; w<10; w++)
      {
        x[w]=random(8);
        y[w]=random(8);
      }
  }  //  ===========  end Rand function ============

int LEDsUp()
{
   rand();               // Get random numbers
   for(z=0; z<255; z++)   // Up Bright - z is LED brightness (0 - 255)
     {
      for (w=0; w<10; w++) // w count 0-9 for x & y arrays
       {
         Rb.setPixelXY(x[w],y[w],z,z,z);
         if (j==0){ return 0; }  // Eliminates delay = 0
         delayMicroseconds(j);
        }
     }
}  //  ============== end LEDsUp function  ==========
int LEDsDown()
{
   rand();               // Get random numbers
   for(z=255; z>0; z--)   // DOWN dim - z is LED dimness (255 - 0)
     {
       for (w=0; w<10; w++) // w count 0-9 for x & y arrays
        {
          Rb.setPixelXY(x[w],y[w],z,z,z);
          if (j==0){ return 0; } // Eliminates delay = 0
          delayMicroseconds(j);
         }
     }
}  //  ============== end LEDsDown function  ==========
```

A video of the working sculpture can be viewed on this web page:
http://www.westgate-works.com/efiles/esculpturev.html