Figure 1: Refraction in a Washbasin

minimize

This program is one that you will modify and use often as you investigate optical phenomena like reflection and refraction. We illustrate its use here by looking at refraction, like the refraction seen in Figure 1, which shows a simple experiment in a washbasin. Figure 2 on page 2 shows a diagram of refraction in a small pool or aquarium. The pool is 200 cm deep and 800 cm long. A fish is located at the bottom of the pool at one end and a person is looking at the fish from the other end. The person's eye is 200 cm above the surface of the water.

Two principles, Fermat's Law and Snell's Law, are helpful in understanding optical phenomena. The program `minimize` is used to apply Fermat's Principle. Fermat's Principle in its first form is almost anthropomorphic. It states that light traveling from one point to another – for example, from the fish to the eye – follows the fastest possible path.

The fastest possible path between two points is normally a straight line but there is a complication – light travels faster (30 cm per nanosecond) in air than it does in water (22.5 cm per nanosecond). Figure 2 shows one possible path light might follow between the fish and the eye. This path is made up of two segments:

- One segment is in air from the eye to a point, denoted $(x, 0)$, on the surface of the water at which the path crosses between air and water. We use the letter x because

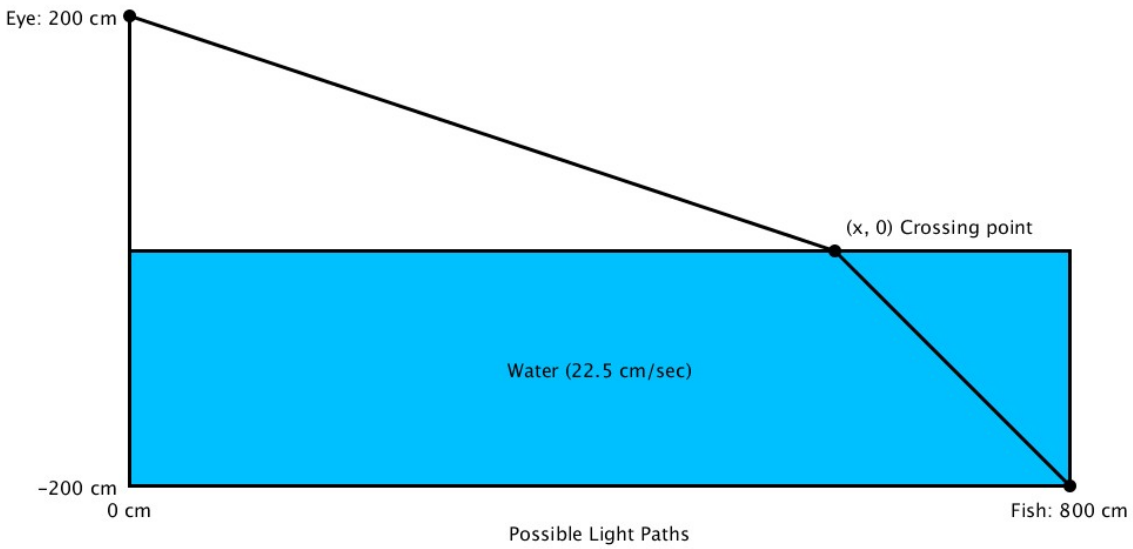

Figure 2: Refraction in a Pool

we don't know the value of this point's x-coordinate. The length of this segment is

$$\sqrt{x^2 + 200^2} \text{ cm}$$

and time required to travel this segment is:

$$\frac{\sqrt{x^2 + 200^2}}{30.0} \text{ nanoseconds.}$$

- The other segment is in water. Its length is:

$$\sqrt{(800 - x)^2 + 200^2} \text{ cm}$$

and the time required to travel this segment is:

$$\frac{\sqrt{(800 - x)^2 + 200^2}}{22.5} \text{ nanoseconds.}$$

Thus, the total time required to travel this path is:

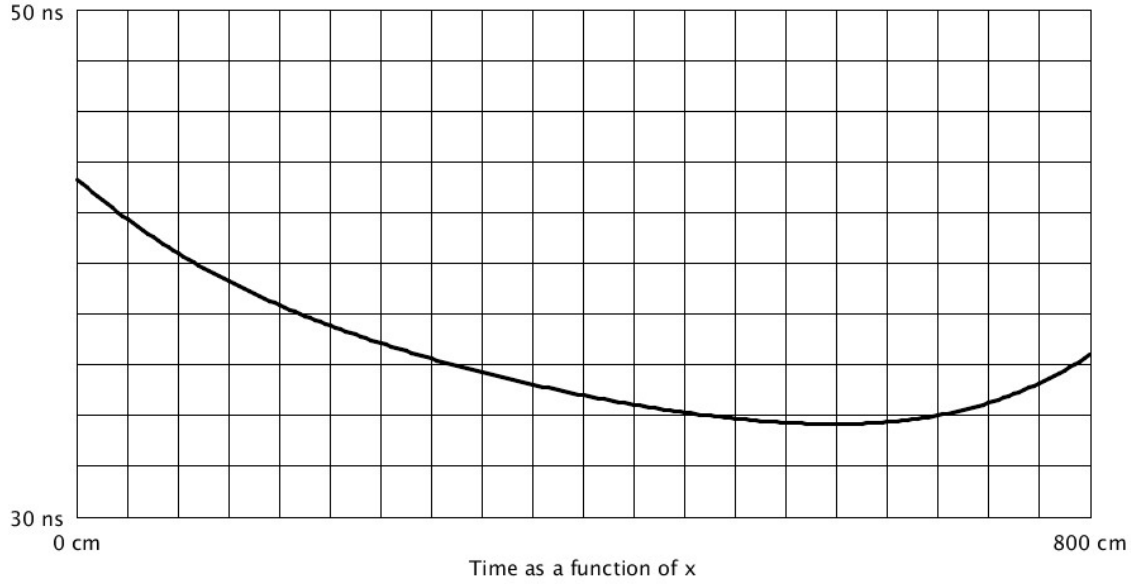

Figure 3: A Function to be Minimized

$$f(x) = \frac{\sqrt{x^2 + 200^2}}{30.0} + \frac{\sqrt{(800 - x)^2 + 200^2}}{22.5}$$

whose graph is shown in Figure 3. To find the actual path light follows we must find the value of x that minimizes this function. That is the purpose of the Processing program `minimize` found below. In addition, this program draws a rough sketch of the graph of this function. You should open and run this program in Processing and look at it in Processing as we discuss it here..

```
float a = 0;                  // The lowest possible value of x.
float b = 800;                // The highest possible value of x.
float yLow;                   // The lowest possible value of y.  Computed in setup.
float yHigh;                  // The highest possible value of y.  Computed in setup.

float fcn(float x)            // The function to be minimized.
{
   return sqrt(pow(x, 2) + pow(200, 2))/30.0 + sqrt(pow(800 - x, 2) + pow(200, 2))/22.5;
}

void settings()               // This routine runs once, before the setup routine.
{
```

```
  size(601, 401);
}

void setup()                  // The setup routine is run once when the program (sketch )starts.
{
  float dx = (b - a)/100;
  float x, y, newX, newY;
  //
  // The next lines find the minimum and maximum possible values of y
  //
  yHigh = fcn(a);
  yLow = fcn(a);
  for(int i = 0; i <= 100; i = i + 1)
  {
    x = a + i * dx;
    yHigh= max(yHigh, fcn(x));
    yLow = min(yLow,fcn(x));
  }
  //
  //  The next lines draw a very rough graph of the function to be minimized.
  //
  background(255);
  x = a;
  y = fcn(a);
  for(int i = 0; i < 100; i = i + 1)
  {
    newX = x + dx;
    newY = fcn(newX);
    line(xScale(x), yScale(y), xScale(newX), yScale(newY));
    x = newX;
    y = newY;
  }
  //
  //   The next lines estimate the minimum value of the function and the corresponding value of x.
  //
  for(int i = 0; i < 8; i = i + 1)     // Use the narrow routine to narrow the interval 8 times.
  {
      narrow();
  }
  print("The fastest path is when x = ");
  println((a + b)/2);
}

void draw()                           // The draw routine is run once for each frame of an animation.
{                                     // This one does nothing but the routine should not be omitted
}

int xScale(float x)                   // This function transforms x values to screen x-coordinates.
{
   return round((x - a) * 600 /(b - a));
```

```
}

int yScale(float y)                    // This function transforms y values to screen y-coordinates.
{
   return  400 - round((y - yLow) * 400/(yHigh - yLow));
}

void narrow()                          // This is the workhorse for minimzing the function.
{
   float dx = (b - a)/20;              // It looks at 21 points in an interval
   float yMin = fcn(a);                // and finds the minimum value of y for those 21 points.
   float xMin = a;
   float x = a;
   for(int i = 0; i < 20; i = i + 1)
   {
      x = x + dx;
      if(fcn(x) < yMin)
      {
        yMin = fcn(x);
        xMin = x;
      }
   }
   a = xMin - dx;                      // Then it narrows the interval by a factor of ten
   b = xMin + dx;                      // including from just left to just right of the found minimum.
}
```

Writing a computer program to find the minimum of any function is actually quite difficult because functions can be quite complicated. The functions we will work with, however, look like Figure 3, U-shaped with a single minimum and these functions are easier to minimize.

The basic idea is simple. We start knowing that the minimum we seek occurs for some value of x in an interval $[a, b]$. For our example the interval is $[0, 800]$. Our plan is to narrow that interval down, effectively trapping the minimum in a smaller interval. We repeat the narrowing procedure several times until we have the minimum trapped in an interval that is extremely small. This gives us a good estimate.

The heart of this procedure is the routine `narrow`. This routine evaluates the function at 21 equally spaced points in the interval $[a, b]$, effectively giving us the partial information shown in Figure 4. The routine `narrow` then looks for the value of x among these 21 points that gives the smallest travel time. In Figure 4 this is $x = 600$. Our new smaller interval is then the interval from the point just to the left of this point to the point just to the right of this point. In Figure 4 this is the interval $[560, 640]$. Notice it is one-tenth the size of the original interval. We repeat this narrowing procedure eight times and use the midpoint of the final interval as our estimate for the value of x that minimizes the function.

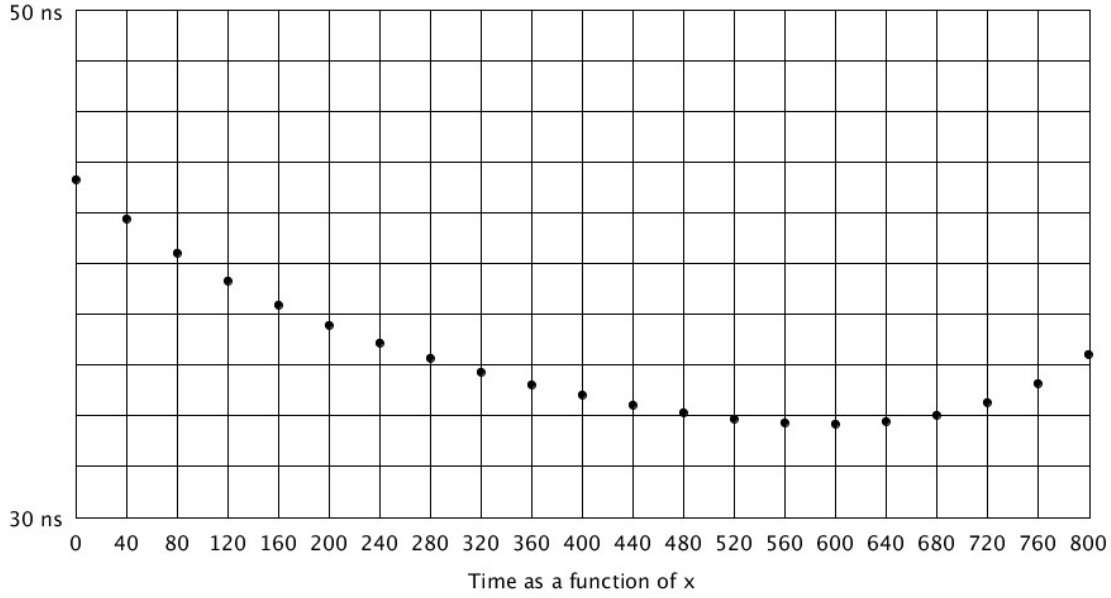

Figure 4: Narrowing the Interval

To modify this program to minimize other functions that come up in the study of optics, replace the definition of `fcn` and the values of `a` and `b`.

Exercises

1. Where would a light ray cross the surface of the water if the fish was only 100 cm below the water's surface?
2. What would a meter stick look like if it rose vertically from a depth of 200 cm?
3. Suppose that a meter stick rose vertically from the point $(0, -200)$. What would it look like to the fish?