Computer Graphics and Multimedia

Project Report on

2D WATER RIPPLE EFFECT

Submitted to Manipal University, Jaipur

Towards the partial fulfilment for the Award of the Degree of

BACHELORS OF TECHNOLOGY

In Computers Science and Engineering

By

Mritunjay Mehta (189301222)

MANIPAL UNIVERSITY
JAIPUR

Under the guidance of

Ms. Bali Devi

Department of Computer Science and Engineering

School of Computing and Information Technology

Manipal University Jaipur

Jaipur, Rajasthan

Index

MOTIVATION & OBJECTIVE

To stimulate ripple effect motion of water on any background.

ALGORITHM – 2D RIPPLE EFFECT

Area Sampling

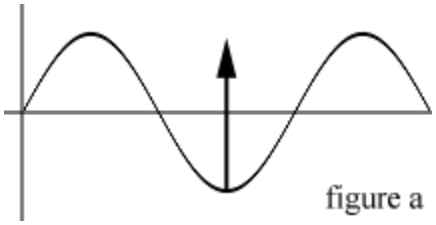
Area sampling is a very common algorithm in computer graphics. Considering a two-dimensional map, the value at (x, y) is affected by values surrounding position (x, y), such as (x+1,y), (x-1,y), (x,y+1) and (x,y-1).

Creating a wave simulation is basically the same, but the value at (x, y) is calculated in a different manner. Earlier mentioned that wave simulation works in three dimensions. Our third dimension is time. In other words: while calculating our wave simulation, we must know how the waves looked like one moment earlier. The resulting map becomes the source map for the next frame.

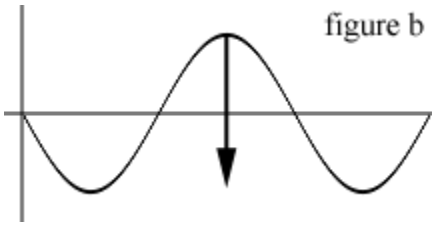
This is the actual wave simulation algorithm:

```
ResultMap[x, y] := (( CurrentSourceMap[x+1, y] +
CurrentSourceMap[x-1, y] +
CurrentSourceMap[x, y+1] +
CurrentSourceMap[x, y-1] ) DIV 2 ) -
PreviousResultMap[x, y]
```

We will notice that the first four values obtained from the current source map are divided by two. This results in a value twice the average. Next, we subtract the value at our working location (x, y) from the previous result map. This produces a new value.



The horizontal gray line represents the average height of the waves. If the previous value at (x, y) was lower than average, then the wave will rise towards average level, as shown in figure a.



If the previous value at (x, y) was higher than average, as shown in figure b, the wave will drop towards average level.

Damping

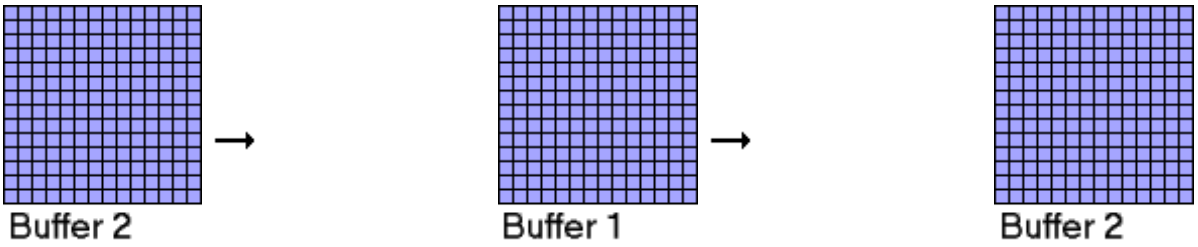
Every time a wave moves up and down, its energy is distributed over a growing area. This means that the amplitude of the wave drops until the wave levels out. Using a damping factor simulates this. The factor, a certain amount or percentage of the amplitude, is subtracted from the current amplitude to let high amplitudes die out fast, and low amplitudes die out slow. In the example below, one sixteenth of the amplitude is subtracted each time it moves.

SO HOW DOES THIS ALGORITHM WORK THEN?

Firstly, we'll need 2 arrays of words (integers) [words, not bytes].

These arrays will hold the state of the water. One holds the current state, the other holds the state from the previous frame. It is important that we have 2 arrays, since

we need to know how the water has changed since the last frame, and the frame before that.



Data from the previous frame (Buffer2) and the frame before that (Buffer1) are used together, and the results written into Buffer1. ***Buffer1*** *contains the* ***previous state*** *of the water.*	*Data from the previous frame (Buffer1) and the frame before that (Buffer2) are used together, and the results written into Buffer2.* ***Buffer2*** *contains the* ***current state*** *of the water.*

```
damping = some non-integer between 0 and 1

 begin loop
    for every non-edge element:
    loop
       Buffer2(x, y) = (Buffer1(x-1,y)
                       Buffer1(x+1,y)
                       Buffer1(x,y+1)
                       Buffer1(x,y-1)) / 2 - Buffer2(x,y)

       Buffer2(x,y) = Buffer2(x,y) * damping
    end loop

    Display Buffer2
    Swap the buffers

 end loop
```

It is also important for the waves to spread out, so the buffers are smoothed every frame.

```
Smoothed(x,y) = (Buffer1(x-1, y) +
                 Buffer1(x+1, y) +
                 Buffer1(x, y-1) +
                 Buffer1(x, y+1)) / 4
```

Now, to combine the two to calculate the new height of the water. The multiplication by two reduces the effect of the velocity.

NewHeight(x,y) = Smoothed(x,y)*2 + Velocity(x,y)

Finally, the ripples must lose energy, so they are damped:

NewHeight(x,y) = NewHeight(x,y) * damping

Note: This can be optimised to:

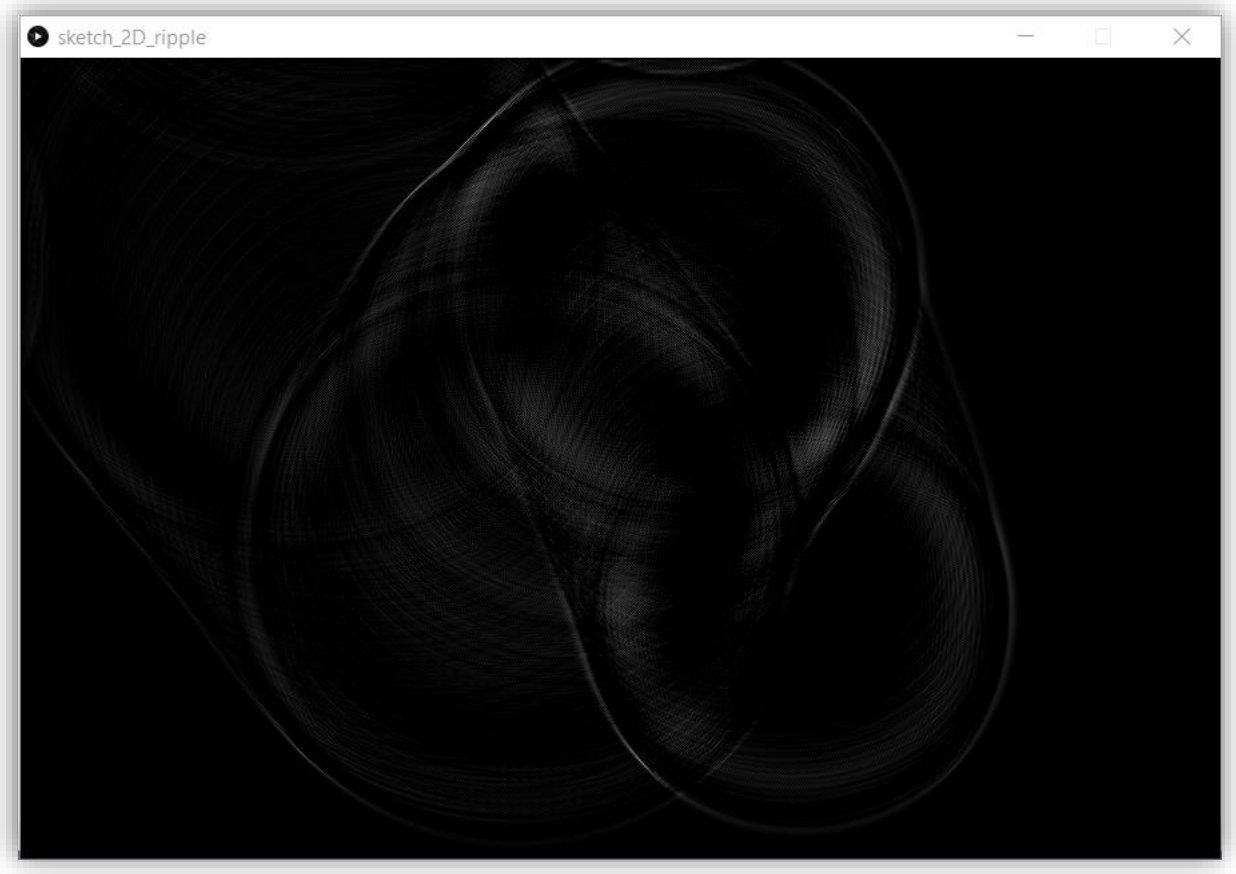
NewHeight(x,y) = NewHeight(x,y) - (NewHeight(x,y)/n)

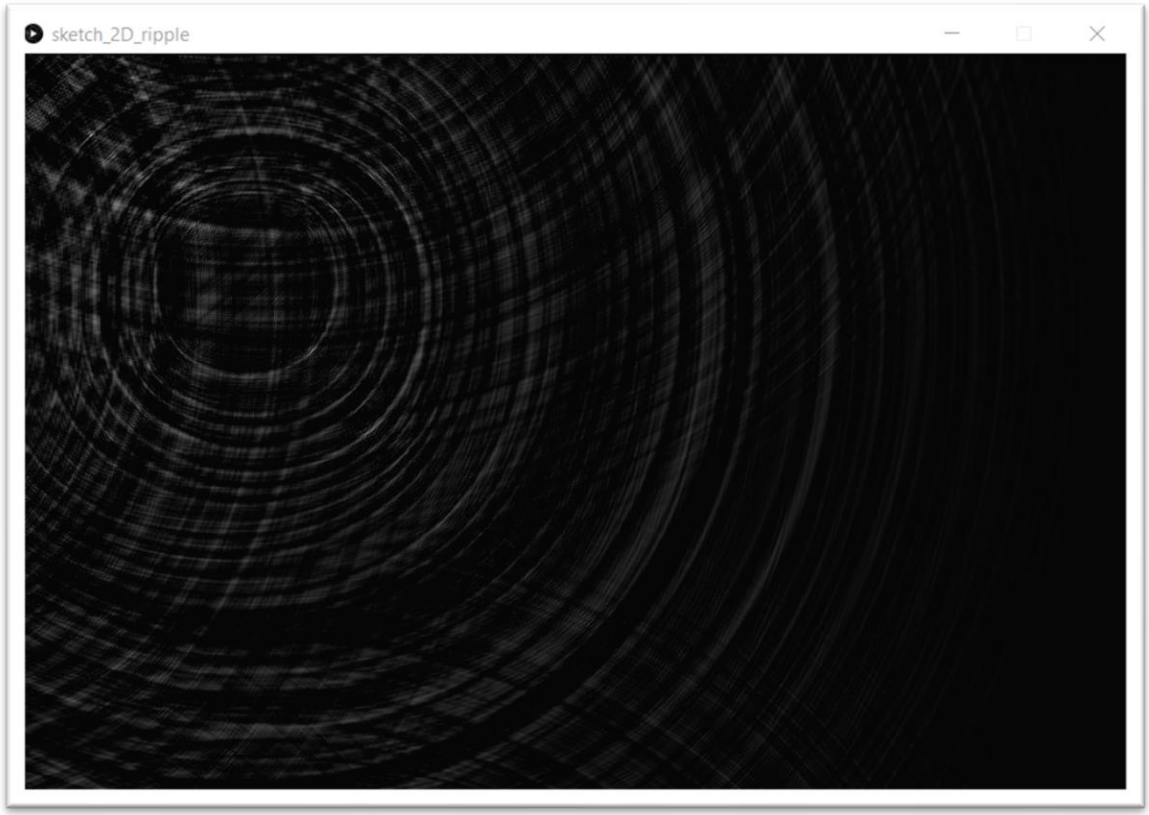
where n is some power of 2.

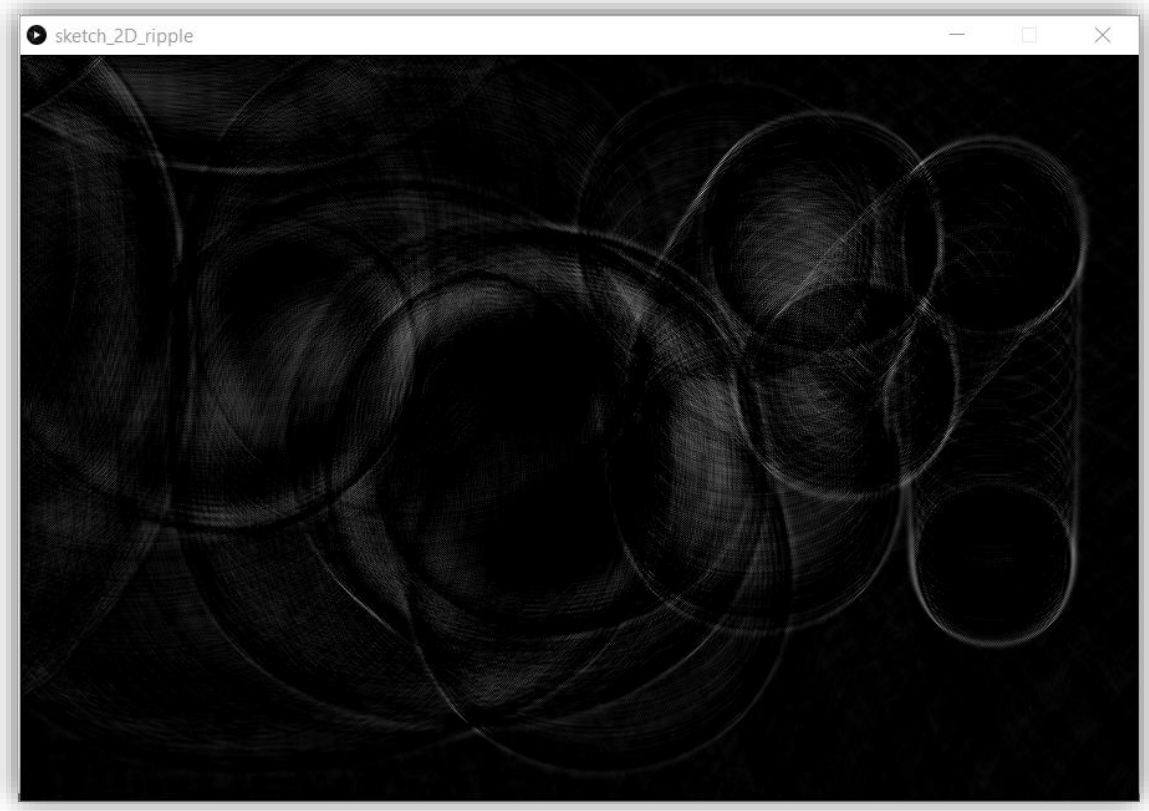
RESULT

A BEAUTIFUL WATER RIPPLE SIMULATION wherever the mouse points or drags.

The simulation is same as how still water behaves when interacted or touched.




sketch_2D_ripple


sketch_2D_ripple

FUTURE SCOPE

This simulation/effect can be further added to any website/webpage/blog/article, etc. to make it more interactive & aesthetic.

REFERENCES

1. 2D Water (archive.org)
2. Microsoft Word - IJITCS-V2-N1-3-CNMT2010-3184 (mecs-press.net)
3. Make a Splash With Dynamic 2D Water Effects (tutsplus.com)
4. The Water Effect Explained - Graphics and GPU Programming - Tutorials - GameDev.net

FINAL CODE

```
int cols;
int rows;
float[][] current;       // = new float[cols][rows];
float[][] previous;      // = new float[cols][rows];

float dampening = 0.995;

void setup() {
  size(900, 600);
  cols = width;
  rows = height;
  current = new float[cols][rows];
  previous = new float[cols][rows];
}

void mouseDragged() {
  previous[mouseX][mouseY] = 1000;
}

void draw() {
  background(0);

  loadPixels();
  for (int i = 1; i < cols-1; i++) {
    for (int j = 1; j < rows-1; j++) {
      current[i][j] = (
        previous[i-1][j] +
        previous[i+1][j] +
        previous[i][j-1] +
        previous[i][j+1]) / 2 -
        current[i][j];
      current[i][j] = current[i][j] * dampening;
      int index = i + j * cols;
```

```
      pixels[index] = color(current[i][j]);
    }
  }
  updatePixels();

  float[][] temp = previous;
  previous = current;
  current = temp;
}
```