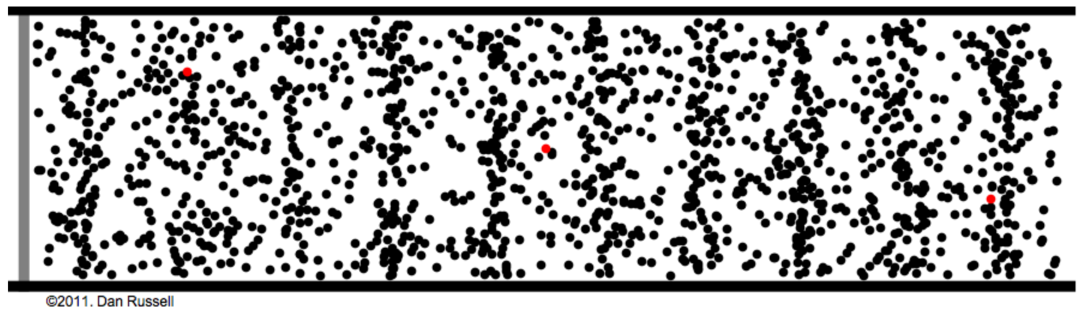
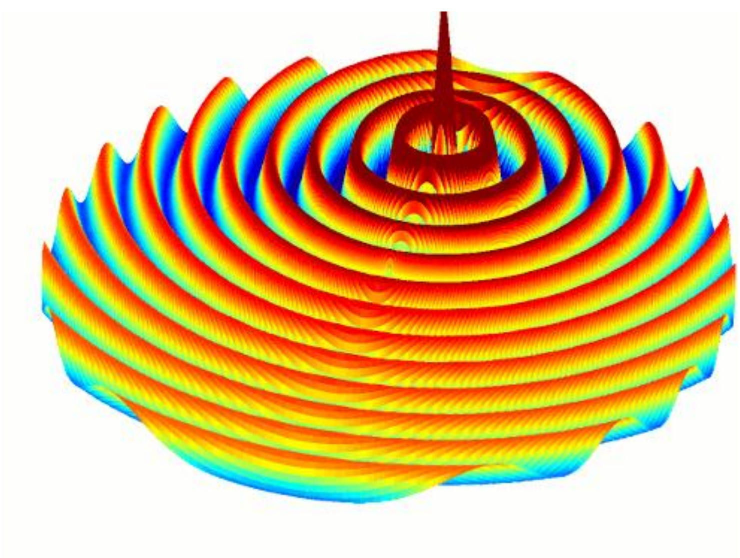
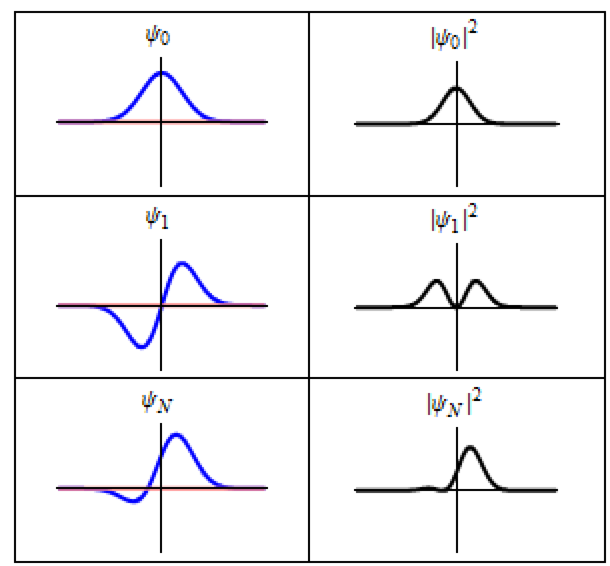
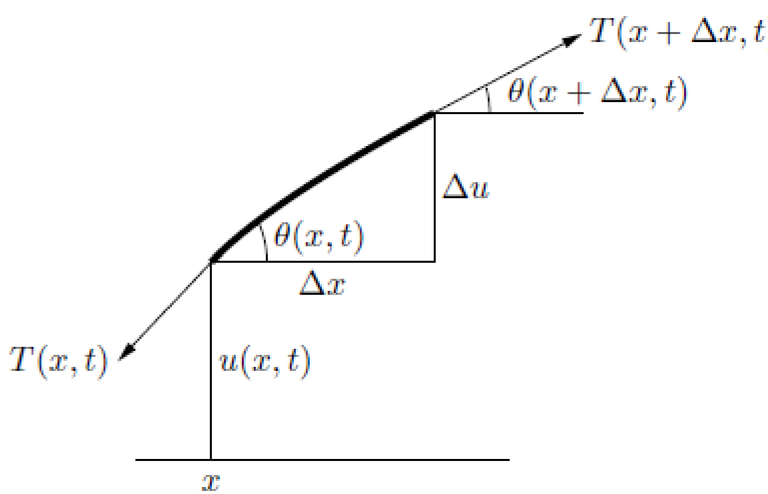
Waves!







Solving something like this…



The Wave Equation

- (1-D)

$$\frac{\partial^2 y}{\partial t^2} = c^2 \frac{\partial^2 y}{\partial x^2}$$

- (n-D)

$$\frac{\partial^2 y}{\partial t^2} = c^2 \nabla^2 y$$

The Wave Equation

$$\frac{\partial}{\partial t}\frac{y_{x,t+1}-y_{x,t}}{\Delta t} = c^2 \frac{\partial}{\partial x}\frac{y_{x+1,t}-y_{x,t}}{\Delta x}$$

$\rightarrow$

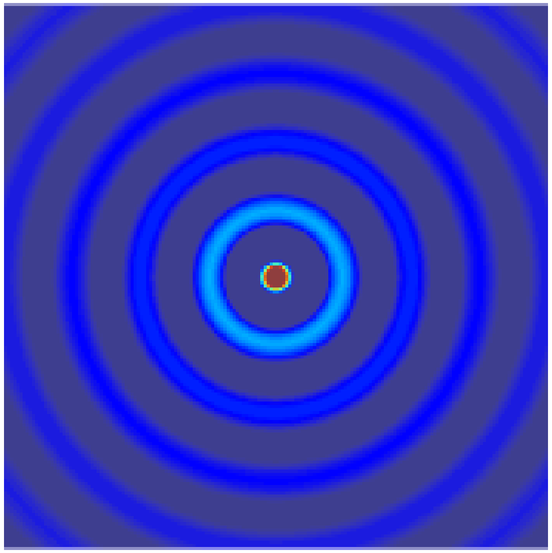
$$\frac{(y_{x,t+1}-y_{x,t})-(y_{x,t}-y_{x,t-1})}{(\Delta t)^2} = c^2 \frac{(y_{x+1,t}-y_{x,t})-(y_{x,t}-y_{x-1,t})}{(\Delta x)^2}$$

$\rightarrow$

$$y_{x,t+1} = 2y_{x,t} - y_{x,t-1} + \left(\frac{c\Delta t}{\Delta x}\right)^2 (y_{x+1,t}-2y_{x,t}+y_{x-1,t})$$

Boundary Conditions

- Examples:
  - Manual motion at an end
    - u(0, t) = f(t)
  - Bounded ends:
    - u(0, t) = u(L, t) = 0 for all t



Discrete solution

- Deal with three states at a time (all positions at t -1, t, t+1)

- Let L = number of nodes (distinct discrete positions)
  - Create a 2D array of size 3*L
  - Denote pointers to where each region begins
  - Cyclically overwrite regions you don't need!

Discrete solution

- Sequential pseudocode:

```
fill data array with initial conditions

for all times t = 0… t_max
    point old_data pointer               where current_data used to be
    point current_data pointer           where new_data used to be
    point new_data pointer               where old_data used to be!
                                         (so that we can overwrite the old!)

    for all positions x = 1…up to number of nodes-2
            calculate f(x, t+1)
    end
    set any boundary conditions!

    (every so often, write results to file)
end
```

CPU Code

```
for (size_t timestepIndex = 0; timestepIndex < numberOfTimesteps;
     ++timestepIndex) {
  if (timestepIndex % (numberOfTimesteps / 10) == 0) {
    printf("Processing timestep %8zu (%5.1f%%)\n",
           timestepIndex, 100 * timestepIndex / float(numberOfTimesteps));
  }

  // nickname displacements
  const float * oldDisplacements =     data[(timestepIndex - 1) % 3];
  const float * currentDisplacements = data[(timestepIndex + 0) % 3];
  float * newDisplacements =           data[(timestepIndex + 1) % 3];

  for (unsigned int a = 1; a <= numberOfNodes - 2; ++a){
      newDisplacements[a] =
              2*currentDisplacements[a] - oldDisplacements[a]
              + courantSquared * (currentDisplacements[a+1]
                      - 2*currentDisplacements[a]
                      + currentDisplacements[a-1]);
  }


  // apply wave boundary condition on the left side, specified above
  const float t = timestepIndex * dt;
  if (omega0 * t < 2 * M_PI) {
    newDisplacements[0] = 0.8 * sin(omega0 * t) + 0.1 * sin(omega1 * t);
  } else {
    newDisplacements[0] = 0;
  }

  // apply y(t) = 0 at the rightmost position
  newDisplacements[numberOfNodes - 1] = 0;
```

GPU Algorithm - Kernel

- (Assume kernel launched at some time t…)

- Calculate y(x, t+1)
  - Each thread handles only a few values of x!
    - Similar to polynomial problem

$$y_{x,t+1} = 2y_{x,t} - y_{x,t-1} + \left(\frac{c\Delta t}{\Delta x}\right)^2 (y_{x+1,t} - 2y_{x,t} + y_{x-1,t})$$

GPU Algorithm – The Wrong Way

- Recall the old “GPU computing instructions”:
  - Setup inputs on the host (CPU-accessible memory)
  - Allocate memory for inputs on the GPU
  - Copy inputs from host to GPU
  - Allocate memory for outputs on the host
  - Allocate memory for outputs on the GPU
  - Start GPU kernel
  - Copy output from GPU to host

GPU Algorithm – The Wrong Way

- Sequential pseudocode:

```
fill data array with initial conditions

for all times t = 0… t_max
        adjust old_data pointer
        adjust current_data pointer
        adjust new_data pointer

        allocate memory on the GPU for old, current, new
        copy old, current data from CPU to GPU

        launch kernel
        copy new data from GPU to CPU
        free GPU memory

        set any boundary conditions!

        (every so often, write results to file)
end
```

GPU Algorithm – The Wrong Way

- Insidious memory transfer!
- Many memory allocations!

GPU Algorithm – The Right Way

- Sequential pseudocode:

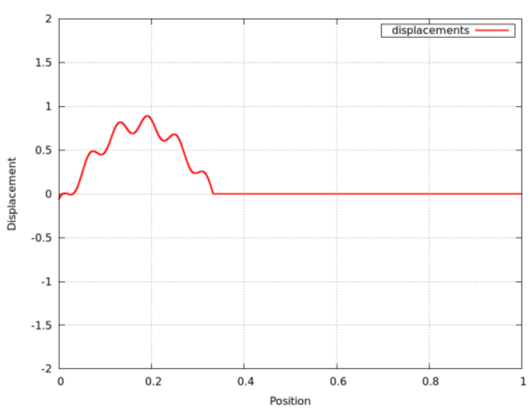
```
allocate memory on the GPU for old, current, new
Either:
        Create initial conditions on CPU, copy to GPU
        Or, calculate and/or set initial conditions on the GPU!

for all times t = 0… t_max
        adjust old_data address
        adjust current_data address
        adjust new_data address

        launch kernel with the above addresses

        Either:
                Set boundary conditions on CPU, copy to GPU
                Or, calculate and/or set boundary conditions on the GPU
End
free GPU memory
```

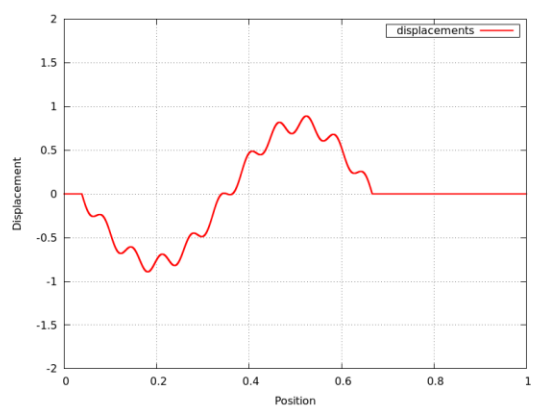
GPU Algorithm – The Right Way

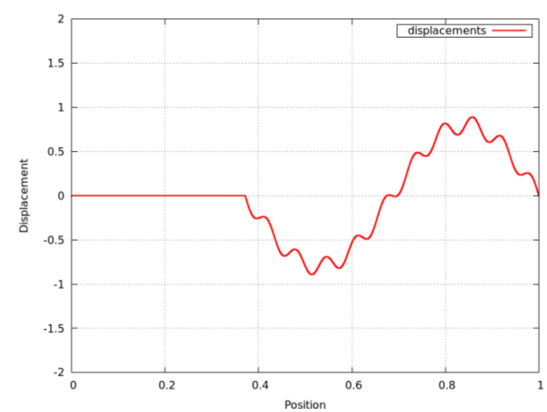
- Everything stays on the GPU all the time!
  - Almost…

Getting output

- What if we want to get a “snapshot” of the simulation?
  – That’s when we GPU-CPU copy!







GPU Algorithm – The Right Way

- Sequential pseudocode:

```
allocate memory on the GPU for old, current, new
Either:
        Create initial conditions on CPU, copy to GPU
        Or, calculate and/or set initial conditions on the GPU!

for all times t = 0… t_max
        adjust old_data address
        adjust current_data address
        adjust new_data address

        launch kernel with the above addresses

        Either:
                Set boundary conditions on CPU, copy to GPU
                Or, calculate and/or set boundary conditions on the GPU
        Every so often, copy from GPU to CPU and write to file
End
free GPU memory
```