Simulating a two dimensional particle in a square quantum box with CUDA

George Zakhour

August 25, 2013

Contents

1 Background, Motive and Experiment

The particle in box problem is one of the first problems given to undergraduate students in a course about quantum mechanics. It showcases how the energy of particles is quantized and it highlights the probabilistic nature of quantum mechanics, especially with the idea that the particle is not located in a fixed position but it has a likelihood of existing at any point in space.

The particle in a box is an experiment in which a particle is stuck inside a box and cannot escape. The energy outside this box is ∞ while inside it is 0. The particle thus moves inside the box of dimensions LxL.

Trying to imagine how these probabilities are scattered in the box is hard and one can't do without a simulation of the physical properties of the particle (position, energy and momentum).

In this simulation I try and simulate the "particle in a box" problem to display the probabilities of the position and the energy of the particle at each state.

2 Expected Output

Searching online, I have found a Youtube video that simulates the particle in a box[1], but lacks essential information on the state of the system, for example the dimensions of the box and the energy levels of the particle at each frame. Although lacking these essential variables I will base my results on the video.

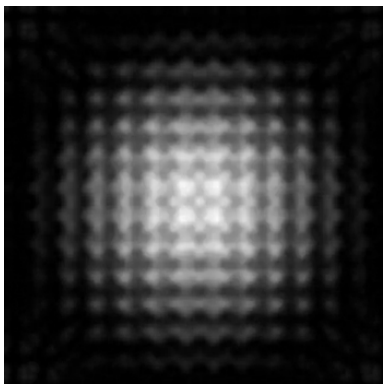


Figure 1: A screenshot of a "particle in a box" simulation found on Youtube

[1] Particle in a Box - Youtube *h*ttp://www.youtube.com/watch?v=jevKmFfcaxE

3 The mathematics of a particle in a box

3.1 In one dimension

3.1.1 Finding the Wave Function from Schrodinger's Equation

The time dependent Schrodinger equation is given as:

$$\left(\frac{-\hbar^2}{2m}\nabla^2 + V\right)\Psi(x,t) = i\hbar\frac{\partial}{\partial t}\Psi(x,t) \tag{1}$$

The time independent Schrodinger equation is given as:

$$\left(\frac{-\hbar^2}{2m}\nabla^2 + V\right)\Phi(x) = E\Phi(x) \tag{2}$$

Where Ψ is the wave equation. In terms of Φ, Ψ is denoted as:

$$\Psi(x,t) = e^{-i(E/\hbar)t}\Phi(x) \tag{3}$$

Assuming the following is a solution to (2):

$$\Phi(x) = A\cos(kx) + B\sin(kx) \tag{4}$$

And given these two conditions that arise from the experiment:

$$\Phi(0) = \Phi(L) = 0$$

We plug them in the (4) and find the following:

$$\begin{cases} A &= 0 \\ k &= \frac{n}{L}\pi \quad \text{(n is the energy level)} \end{cases} \tag{5}$$

To find the value of B we need to normalize the equation.
The probability of finding the particle inside $[0; L]$ is 1 because it cannot escape.

$$\int_0^L |\Phi|^2 dx = 1 \tag{6}$$

$$B^2\int_0^L \sin^2\left(\frac{n}{L}\pi x\right)dx = 1 \tag{7}$$

And thus $B = \sqrt{\frac{2}{L}}$.
Finally we get the solution to the time independent Schrodinger equation

$$\Phi(x) = \sqrt{\frac{2}{L}}\sin(\frac{n}{L}\pi x) \tag{8}$$

3.1.2 Energy at each quantum level n

We note the following

$$\frac{\partial^2}{\partial x^2}\Phi = -\left(\frac{n\pi}{L}\right)^2\Phi \tag{9}$$

If we replace the results in (2) we find:

$$E_n = \frac{\hbar^2 n^2 \pi^2}{2mL^2}$$

Or simply:

$$E_n = n^2 E_1$$
$$E_1 = \frac{\hbar^2 \pi^2}{2mL^2}$$

3.1.3 Probability function of the position

The probability of finding the particle between a and b is the following:

$$\begin{cases} \int_a^b |\Phi|^2 dx & \text{if } a, b \in [0, L] \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

If we wish to find the position inside a box of width ϵ and center a we would integrate between $a - \epsilon/2$ and $a + \epsilon/2$ where both ends are between 0 and L The integration will lead to the following:

$$P(x) = \frac{\epsilon}{L} + \frac{1}{2n\pi}\left[\sin\left(\frac{2n\pi}{L}(a - \epsilon/2)\right) - \sin\left(\frac{2n\pi}{L}(a + \epsilon/2)\right)\right]$$

Which can be reduced more to the following form:

$$P(x) = \frac{\epsilon}{L} - \frac{1}{n\pi}\sin\left(\frac{n\pi}{L}\epsilon\right)\cos\left(\frac{2n\pi}{L}x\right)$$

3.2 In two dimensions

3.2.1 The time-independent solution

This time we suppose the solution is

$$\Phi(x, y) = X(x)Y(y) \tag{11}$$

$$X(x) = A\cos(k_x x) + B\sin(k_x x)$$
$$Y(y) = C\cos(k_y y) + D\sin(k_y y)$$

Plugging (11) in (2) and doing similar operations as *3.1.1* we get the following solution:

$$\Phi(x, y) = \frac{2}{L}\sin\left(\frac{n_x\pi}{L}x\right)\sin\left(\frac{n_y\pi}{L}y\right) \tag{12}$$

3.2.2 Energy in two dimensions

Doing the same steps as *3.1.2* we find that the energy has become:

$$E_{n_x,n_y} = \frac{\hbar^2\pi^2}{2mL^2}(n_x^2 + n_y^2) \tag{13}$$

Or simply:

$$E_{n_x,n_y} = (n_x^2 + n_y^2)\ E_{1,1}$$
$$E_{1,1} = \frac{\hbar^2\pi^2}{2mL^2}$$

3.2.3 Probability function of the position

Similar to section *3.1.3* to find the probability of finding the particle inside the box of size $\epsilon x \epsilon$ we integrate the wave function inside a box of center (x, y) and width ϵ.

$$P(x,y) = \int_{y-\epsilon/2}^{y+\epsilon/2} \int_{x-\epsilon/2}^{x+\epsilon/2} |\Phi(x,y)|^2 dxdy$$

After evaluating the integral we get the following function:

$$P(x,y) = \frac{1}{L^2}\, p(x)\, p(y) \tag{14}$$

$$p(\alpha) = \epsilon - \frac{L}{n_\alpha \pi} \cos\left(\frac{2n_\alpha \pi}{L}\alpha\right) \sin\left(\frac{n_\alpha \pi}{L}\epsilon\right)$$

3.3 Generalizing the time-independent solution

The equation that we have found so far depends on two quantum numbers n_x and n_y and describes the particle for these energy levels only. However the final time-dependant equation is a combination of many of these equations. The final time-independent equation is:

$$\Phi(x,y) = \sum_n c_n \Phi_{n_x,n_y}(x,y) \tag{15}$$

Where the constant c_n is the square root of the probability of getting the equation Φ_{n_x,n_y} that describes the particle in the box.

3.4 Generalizing the probability function

Since we have a more general wave function we need to have a more general probability function for the position. The new definition is generated from integrating Φ between $a - \epsilon/2$ and $a + \epsilon/2$. The result is:

$$P(x,y) = \sum_n c_n^2\, P_n(x,y)$$

4 Implementation

4.1 Synopsis

This simulation will display the probability of finding the particle in sub-boxes in the box as well as display the energy in the sub-box of width ϵ. Each frame displays the particle with a new set of probabilities for each energy level.

The probability of the position will be visualized by the intensity of the color. The brighter the color, the higher the probability.

The energy however will be visualized using the standard colors assigned to energies; lower energies have blue-shades while high energies have red-shades.

4.2 Conventions

Some of the conventions adopted throughout the code are

- words in functions, structs and variable names are separated by an `_` (underscore)
- CUDA kernels start with the keyword `cuda_`.
  For example `cuda_probability`
- CUDA device functions start with `cuda_` and end with `_device`.
  For example `cuda_probability_1d_device`
- Tiny mathematical and physical constants, such as $\hbar$ are expressed as float numbers without their orders.
  For example $\hbar = 1.054 \cdot 10^{-34}$ is defined as `#define HBAR 1.054`

4.3 Brightness and Color mapping

4.3.1 Brightness

The intensity of the colors denote the probability of finding the particle. Since probabilities are always between 0 and 1, converting them to intensities is just a matter of mapping the probabilities from $[0, 1]$ to $[0, 255]$.

However ϵ is a small number, so the probability is always small; usually less than 0.1%. Therefore we need to find the highest probability in the space and remap from $[0, \max]$ to $[0, 255]$. At first I have tried to brute force the problem exploiting my GPU using functions to find maximums using reduction. But the process of finding the highest probability of a particle with 10 energy levels inside a box divided into 262,144 boxes took around 28ms. Allowing me to get only 35fps in the animation.
A new solution needed to be adopted and I went over the equations again to try and find that maximum analytically. My results are:

For $p(\alpha)$ to be maximal

$$\frac{\epsilon}{L} - \frac{1}{n_\alpha \pi} \sin\left(\frac{n_\alpha \pi}{L}\epsilon\right) \cos\left(2\frac{n_\alpha \pi}{L}\alpha\right)$$

needs to be maximal. And this is achieved when $\cos x = -1$ which is possible since $0 \leq x \leq 2\pi$. Therefore the highest probability for one energy level is:

$$\frac{\epsilon}{L} + \frac{1}{n_\alpha \pi} \sin(\frac{n_\alpha \pi}{L} \epsilon)$$

Following similar analysis we can deduce that the highest probability (denoted by m) is close to, but not exactly:

$$m = m_x . m_y$$

$$m_\alpha = \frac{\epsilon}{L} + \sum_i \left(c_i^2 \frac{1}{n_{\alpha_i} \pi} \sin\left(\frac{n_{\alpha_i} \pi}{L} \epsilon \right) \right)$$

UPDATE After implementing and testing the algorithm the uncertainty in the algorithm has proved to be overwhelming and the probabilities' range was no longer $[0, 1]$ but much smaller. Therefore a fallback was needed.

The approach followed was a bruteforce solution where we compute the probabilities at each at each pixel and search for the largest. However to make the searching faster a reduction algorithm was adopted to speed it up. The algorithm creates a much smaller array in which it stores the maximum of a chunk of the original array. Then on the CPU we loop over the results (which are usually 32) and pick the maximum of these. The following illustration describes how the maximum reduction algorithm works The fact that this algorithm was used

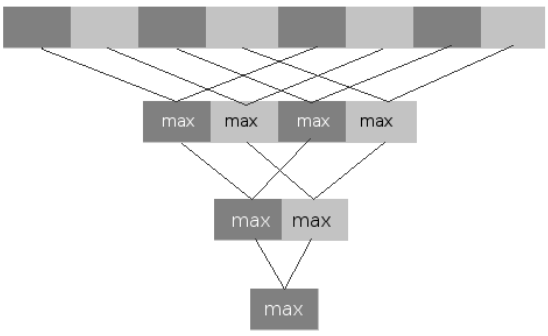


Figure 2: Finding the maximum element in an array using reduction

adds a restriction on the dimensions of the window. The height and the width of the window need to be a power of 2 for this algorithm to work effectively. No solution has been made up yet to generalize the dimensions of the window.

4.3.2 Tint and Color

The tint in the color represents the energy of the particle. Red is the highest energy and blue is the lowest. If we can map the energy in an interval between 0

and 1 we can easily get the RGB values. The following equation can be applied

$$\begin{bmatrix} R_{xy} \\ G_{xy} \\ B_{xy} \end{bmatrix} = P(x,y) \begin{bmatrix} 255E \\ 20 \\ 255(1-E) \end{bmatrix}$$

Where $P(x,y)$ is the probability of finding the particle at (x,y) and between 0 and 1. And E is the energy remapped between 0 and 1.

To problem of remapping the energy is easy to solve. Through mathematical analysis we can show that the highest energy we can find is

$$E_{max} = (n_{max_x}^2 + n_{max_y}^2)\frac{\hbar^2\pi^2}{2mL^2}$$

And therefore we can remap any energy we find to a value in the interval $[0,1]$ and find the corresponding RGB values.

4.4 Next set of probabilities

Every frame in the animation consists of a new set of probabilities close to the ones of the previous frame. A problem came up and that is how to generate new set of probabilities close to the ones in the previous frame to insure the smooth animation between the frames.

The model adopted to insure the smoothness and the continuity in the probabilities consists of multiple sine waves where the period of one is 10 times more than its neighbor and so on. If plotted, the model looks like the following: (the sines were stretched vertically for a better visualization) After finding the values

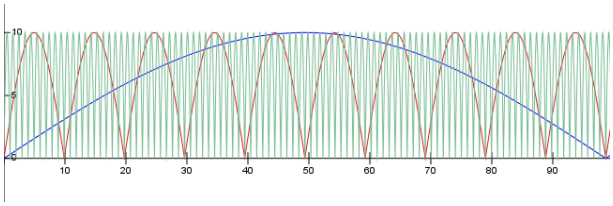


Figure 3: Plotting $x(t) = 10\left|sin\left(\frac{0.1}{\pi}t\right)\right|$, $y(t) = 10\left|sin\left(\frac{1}{\pi}t\right)\right|$, $z(t) = 10\left|sin\left(\frac{10}{\pi}t\right)\right|$

of each sine wave at a time t we can normalize the answers so that their sum is 1. Bellow is a table representing some values normalized.

Time	x	y	z
0.1	0.0091	0.0915	0.8994
0.2	0.0096	0.0957	0.8947
0.3	0.0104	0.1035	0.8861
0.4	0.0116	0.1159	0.8725
0.5	0.0136	0.1350	0.8515
0.6	0.0166	0.1648	0.8186
0.7	0.0215	0.2135	0.7649
0.8	0.0304	0.3006	0.6690
0.9	0.0490	0.4834	0.4675
1.0	0.0824	0.8102	0.1074
1.1	0.0479	0.4698	0.4822
1.2	0.0368	0.3590	0.6042
1.3	0.0322	0.3134	0.6544
1.4	0.0309	0.2987	0.6704
1.5	0.0317	0.3053	0.6630
1.6	0.0347	0.3324	0.6329
1.7	0.0405	0.3859	0.5736
1.8	0.0509	0.4818	0.4673
1.9	0.0701	0.6595	0.2704
2.0	0.0859	0.8022	0.1119

Table 1: Values from $x(t)$, $y(t)$ and $z(t)$ at different times

A general formula can be deduced to find the equation of the nth wave.

$$P_n(t) = \left| sin\left(\frac{10^{1-n}}{\pi}t\right)\right| \tag{16}$$

4.5 (G)UI

The GUI will be farely simple, a window where the simulation is drawn on. The user can control the simulation by keystrokes listed bellow.

.	Increase the time offset
,	Decrease the time offset
SPACE	Toggle pausing
0	Reset the animate (t = 0)
ESC	Quit the simulation

As well, the user can control the number of wave functions that are being simulated using command line arguments as follow

```
pbox n
```

Where `n` is the number of wave functions the user wishes to simulate.

5 API

5.1 Structures

5.1.1 particle

Members

- `float mass`

  The mass of the particle.

- `int energy_levels`

  The number of energy levels the particle has, i.e. the number of wave functions that describe the particle.

- `float *particles`

  The probabilities of each wave function. The length of the array this variable points to should be `energy_levels`.

5.2 Global Variables

The program uses these global variables

- `cudaGraphicsResource *resource`
  - The graphics resource that is linked to the OpenGL environment.
- `GLuint buffer`
  - The buffer used by OpenGL.
- `float INCREASE_TIME`
  - The time offset used to increase the time between the frames.
- `particle p`
  - The particle that the program is about.
- `float t`
  - The current time (set as the offset).
- `int frames`
  - The number of frames.
- `float total_time`
  - The total time.
- `int PAUSE`
  - Whether the animation is paused.

5.3 Functions

5.3.1 float max(float *numbers, int N)

This function finds the maximum number in an array of numbers using reduction as described in 4.3.1.

Parameters
`float *numbers`
- A pointer of the array to find the maximum of.

`int N`
- The length of `numbers`. **Should be a power of 2**.

Returns
`float`, the maximum number in the array.

5.3.2 void next_probabilities(float t, int N, float *probabilities)

This function generates a new set of probabilities as described in 4.4.

Parameters
`float t`
- The time parameter required in the algorithm.

`int N`
- the number of parameters to generate, i.e. the length of the next parameter.

`float *probabilities`
- The array to write the new probabilities to.

5.3.3 void create_particle(particle *p, int N, float mass)

This function creates a new particle.

Parameters
`particle *p`
- The particle to store the new particle in.

`int N`
- The number of wave functions (energy levels) that describe the particle.

`float mass`
- The mass of the particle

5.3.4 float probability(particle *p, float x, float y)

This function finds the probability of finding a particle inside a box of center (x, y) and of width and height ϵ.

Parameters
`particle *p`
- The particle to act on

`float x`
- The x-coordinate of the particle

`float y`
- The y-coordinate of the particle

Returns
`float`, the probability of finding the particle at (x, y)

5.3.5 float max_probability(particle *p)

This function finds the highest probability of existing at a position (x, y) in the space, i.e. inside the box. The function is used to later on map the probabilities from $[0, \max]$ to $[0, 255]$ for color-brightness purposes. The algorithm adopted is described in 4.3.1.

Parameters
`particle *p`
- The particle to find the highest probability of existing

Returns
`float`, the highest probability to exist.

5.3.6 void initGL(int *argc, char **argv)

This function initializes the OpenGL environment.

Parameters
`int *argc`
- The length of the next parameter.

`char **argv`
- The parameters supplied to the OpenGL environment.

5.3.7 void display()

This function takes care of drawing the output on the canvas on every iteration. As well it prints out the current time, the average time per frame in milliseconds

and the offset that is used to increase the time.

5.3.8 void key(unsigned char k, int x, int y)

This function manages the keystrokes in the canvas.

Parameters
`unsigned char k`
- The character pressed.

`int x`
- The x-coordinate where the character is pressed.

`int y`
- The y-coordinate where the character is pressed.

5.3.9 void free_resources()

This function is used to clean after closing the window, i.e. free the resources.

5.3.10 void createVBO(GLUint *buffer, cudaGraphicsResource **resource, unsigned int flags)

This function initializes the buffer and the resources that are used by OpenGL.

Parameters
`GLuint *buffer`
- The buffer used by OpenGL.

`cudaGraphicsResource **resource`
- The CUDA resource to link to the buffer

`unsigned int flags`
- The flags used by OpenGL.

5.3.11 void launch_kernel(uchar4 *pos)

This function launches the kernel that fills the CUDA resource which will be used to draw on the canvas.

Parameters
`uchar4 *pos`
- The array of pixel data

5.3.12 void runCuda(cudaGraphicsResource **resource)

This function creates the resources for the kernel and launches it.

Parameters
`cudaGraphicsResource **resource`
- The CUDA resource.

5.3.13 void run(int argc, char **argv)

This function runs everything, i.e. initializes the environment, selects a valid GPU card, creates the resources and launches the GUI.

Parameters
`int argc` the length of the next parameter.
`char **argc` the parameters used by the OpenGL environment.

5.3.14 void usage(char* program_name)

This function prints out the help message.

Parameters
`char* program_name`
- The name of the program to run

5.4 CUDA Kernels

5.4.1 __global__ void cuda_max(float *numbers, int N, float *partialMax)

This kernel finds the maximum in buckets (sub-array) of the numbers array and fills the partialMax array using the reduction method described in 4.3.1.

Parameters

`float *numbers`
- The numbers to find the maximum of.

`int N`
- The length of the array of numbers.

`float *partialMax`
- The list containing the maximum of the buckets.

5.4.2 __global__ void cuda_probability_to_map(float *probabilities, int n, float *map)

This kernel maps the coordinate array to the probability array.

Parameters

`float *probabilities`
- The probability set of each wave function.

`int n`
- The number of wave functions.

`float *map`
- The array that will be filled with probabilities

5.4.3 __global__ void cuda_probability(float *p, in N, float x, float y, float *probability)

This kernel finds the probability of finding the particle in a certain position.

Parameters

`float *p`
- The probability set of each wave function.

`int N`
- The number of energy levels.

`float x`
- The x-coordinate of the particle.

`float y`
- The y-coordinate of the particle.

`float *probability`
- The variable to write the probability to.

5.4.4 __global__ void kernel(uchar4 *ptr, float *probabilities, int N, float max_proba)

This kernel fills the pixel array with the corresponding colors to display them.

Parameters

`uchar4 *ptr`
- The array of pixels

`float *probabilities`
- The array of probabilities of each wave function.

`int N`
- The number of wave functions.

`gloat max_proba`
- The highest probability in the space.

5.5 CUDA Device Functions

5.5.1 __device__ float cuda_probability_1d_device(int n, float x)

This function finds the probability of finding the particle in one dimension at the position x and at the energy level n.

Parameters
`int n`
- The energy level of the particle

`float x`
- The position of the particle

Returns
`float`, the probability.

5.5.2 __device__ float cuda_probability_1d_device(float *probabilities, int n, float x, float y)

This function finds the probability of finding the particle at a fixed position given a set of probabilities, the number of energy levels and the position.

Parameters
`float *probability`
- The probability of each energy level.

`int n`
- The number of energy levels.

`float x`
- The x-coordinate of the particle.

`float y`
- The y-coordinate of the particle.

Returns
`float`, the probability.

5.5.3 __device__ float energy(float mass, int n)

This function finds the energy of the particle at a precise energy level.

Parameters
`float mass`
- The mass of the particle.

`int n`
- The energy level the particle is at.

Returns
`float`, the energy at the energy level n.

5.5.4 __device float highest_energy(float mass, int n)

This function finds the highest energy the particle can reach. This function is used for color mapping described in 4.3.2.

Parameters
`float mass`
- The mass of the particle.

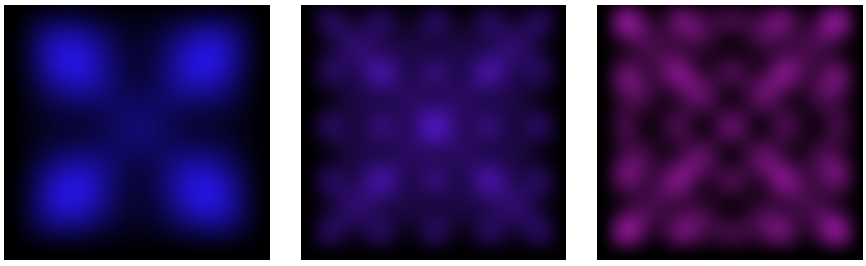
`int n`
- The highest energy level the particle can reach.

Returns
`float`, the highest energy

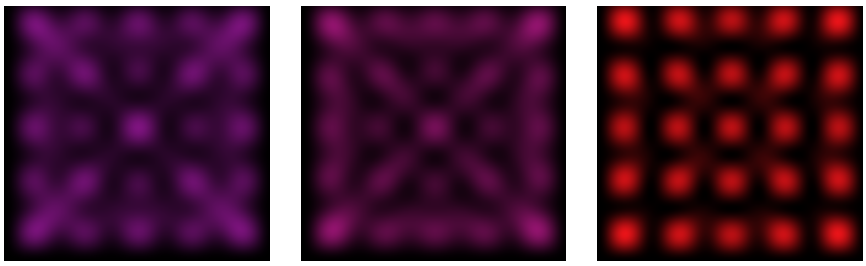
6 Results

6.1 Screenshots

These are some screenshots taken from the simulation. Only 5 wave functions are illustrated.



1 2 3



4 5 6

From observing the screenshots we can generate a table that encapsulate the wave functions most visible in the screenshots

Screenshot	1	2	3	4	5
1	×	×			
2	×	×	×		×
3		×	×	×	
4		×	×		×
5			×		×
6	×				×

6.2 Benefits

Thanks to this simulation the problem of the Particle in a box is now much clearer to me, and I was able to understand more about the topic.

As well it helped me visualize another problem related and that is *Heisenberg's Uncertainty Principle* described mathematically as $\delta_x \delta_p \geq \frac{\hbar}{2}$. As described in section 4.3.1 the more intense the color at a pixel is, the higher the probability.

For higher the quantum number is the larger the energy, and since the energy is directly related to the momentum of the particle we should expect a large uncertainty in the position of the particle. That is perfectly illustrated in this simulation, as the graphics become red the higher the energy is and we can see numerous bright spots on the canvas, meaning there is a high probability for the particle to exist in many positions, therefore agreeing with Heisenberg's Uncertainty Principle.

7 Code License: GNU General Public License