LU3PY002 – Physique Numérique
Licence 3 de physique

Compte rendu du Projet

Thomas Le Varlet
Diane-Laure Dagot
Mono 3

Introduction

A slower speed of light (1), programmé par des chercheurs du MIT, est un jeu permettant de mieux visualiser la relativité restreinte. Nous avons choisi de programmer un jeu mettant en avant l'aspect probabiliste de la physique quantique. On s'intéresse au mouvement d'une particule et à son énergie lorsque celle-ci rencontre une barrière de potentiel de hauteur U. Le but du jeu est de maximiser l'énergie cinétique de notre électron afin d'optimiser nos chances de traverser la barrière.

Méthodologie

Physique sous-jacente

On considère un électron qui rencontre une barrière de potentiel de largeur L et de hauteur V. L'électron peut être soit réfléchi, soit transmis. Au cours de ce processus, son énergie E reste constante. L'état quantique de l'électron est alors décrit par une fonction d'onde solution de l'équation de Schrödinger indépendante du temps :

$$-\frac{\hbar^2}{2m}\frac{\mathrm{d}^2\Psi}{\mathrm{d}x^2} + V(x)\Psi(x) = E\Psi(x)$$

avec V(x) le potentiel qu'il subit par l'électron, E son énergie, m sa masse et ħ la constante de Planck.

On se place dans le cas où l'énergie de la particule est inférieure à la hauteur de la barrière :

0 < E < V0 :

La particule a une probabilité non nulle de traverser la barrière de potentiel, bien que son énergie soit inférieure à la hauteur de celle-ci. Sa fonction d'onde dans cette barrière n'est pas nulle mais a le comportement d'une onde évanescente.

$$\varphi''(x) - \frac{2m(V_0 - E)}{\hbar^2}\varphi(x) = 0$$

En utilisant la continuité de la fonction d'onde et de sa dérivée, on peut ensuite déterminer le coefficient de transmission :

$$T(E) = \frac{1}{1 + \frac{V_0^2}{4E(V_0 - E)} \sinh^2\left(\frac{a}{\hbar}\sqrt{2m(V_0 - E)}\right)}$$

avec « a » la largeur de la barrière et « m » la masse de l'électron.

On constate donc que plus la barrière est large, plus la transmission d'un électron avec une énergie donnée sera faible. De même, lorsque la hauteur de la barrière augmente, le coefficient de transmission diminue, il faut donc fournir plus d'énergie à l'électron pour qu'il puisse traverser la barrière.

On se place maintenant dans le cas où l'énergie de la particule est supérieur à la hauteur de la barrière :

E > V0 :

Dans la barrière, on a une équation de Schrödinger :

$$\varphi''(x) + \frac{2m(E - V_0)}{\hbar^2}\varphi(x) = 0$$

On en déduit le coefficient de transmission suivant :

$$T = \frac{4E(E - V_0)}{4E(E - V_0) + V_0^2 \sin^2\left[\sqrt{\frac{2m(E - V_0)}{\hbar}}l\right]}$$

avec « l » la largeur de la barrière et « m » la masse de l'électron. Si l'electron n'est pas transmis, il est reflechi.

Règles du jeu

On contrôle la position verticale de la probabilité de présence d'un électron (symbolisée par un bleu de plus en plus contrasté lorsqu'on s'éloigne du rayon) avec les touches haut et bas du clavier dans le but de collisionner les photons jaunes. Au départ, l'électron possède une énergie cinétique nulle. Pour l'augmenter, il faut collisionner avec un photon qui arrive de la gauche de l'écran, avec une hauteur aléatoire (figure 1).

Plus l’électron a une longue durée de contact avec le photon, plus son énergie cinétique augmentera. L’énergie cinétique maximale que peut avoir l’électron est de 1 eV, il en est de même pour le potentiel maximal de la barrière, bornée de 0 à 1 eV.

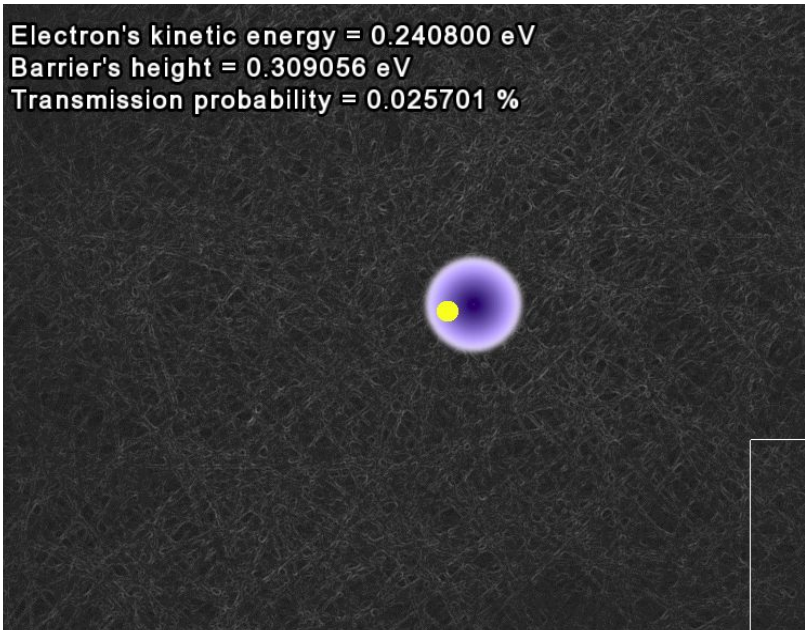

Figure 1 : Collision entre le photon et l'électron

Une barrière de potentiel arrive de la droite avec une hauteur U et une largeur d aléatoires. Lorsque l’électron se trouve dans la zone de la barrière (figure 2), son énergie cinétique devient constante et il est alors impossible de contrôler son déplacement vertical. De plus, il se placera automatiquement à la hauteur correspondant à son énergie par rapport à un axe vertical d'énergie (invisible) que suit également la barrière de potentiel.

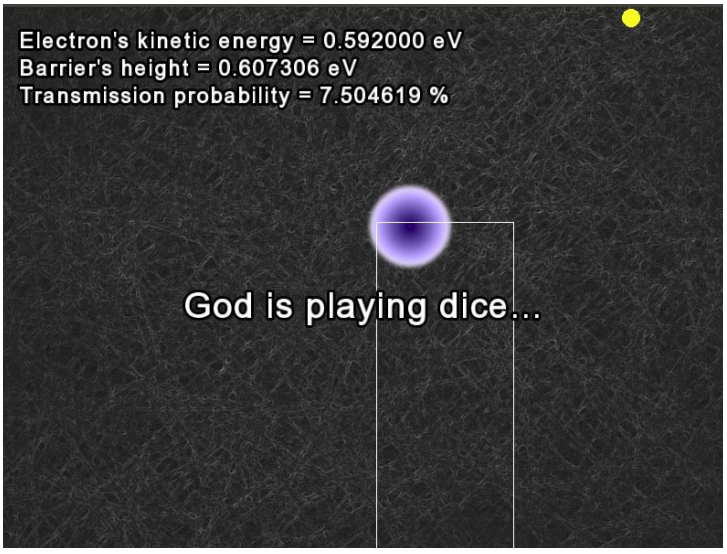

Figure 2 : Electron tentant de traverser la barrière de potentiel avec une référence à une citation d’Albert Einstein (2)

Le but du jeu est d'avoir une énergie cinétique de l'électron suffisamment élevée par rapport à celle de la barrière pour optimiser ses chances de la traverser et de continuer le jeu (figure 3). Conformément aux lois de la mécanique quantique, une énergie cinétique de l'électron supérieure ou inférieure à la barrière n'assure pas la transmission ou la reflection.

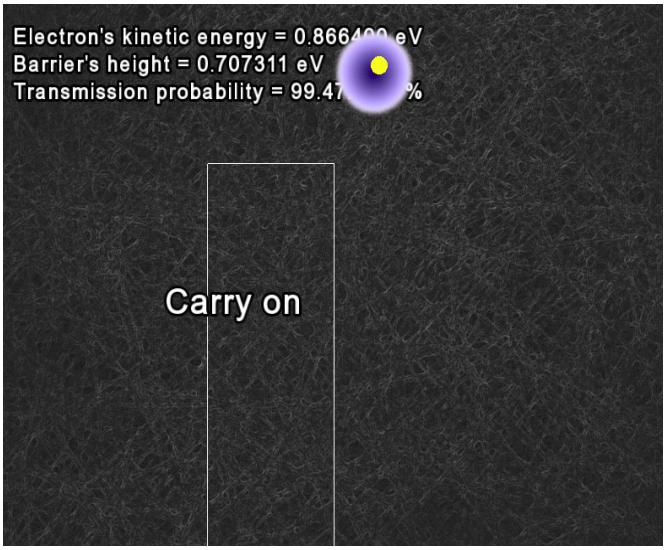

Figure 3 : Electron transmis

Lorsque l'électron a été réfléchi (figure 4), le joueur peut tapper sur la barre espace du clavier pour recommencer avec un autre puits de départ.

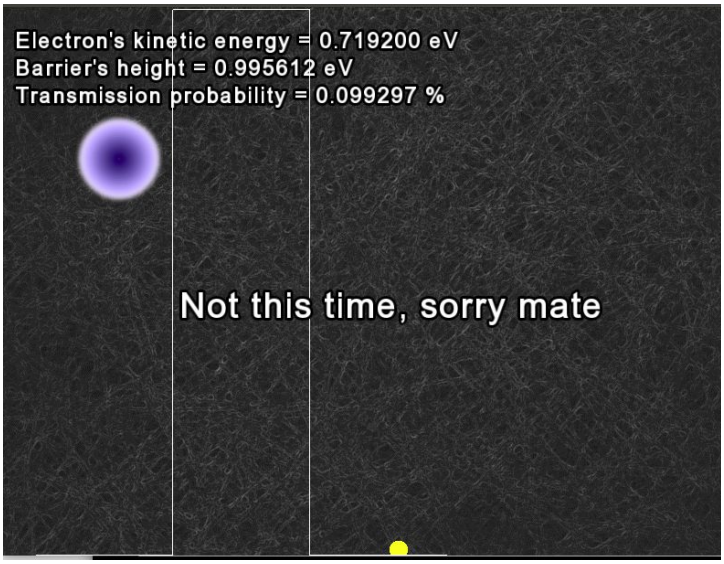

Figure 4 : Electron reflechi

L'énergie de l'électron diminue constamment lorsqu'il se trouve en dehors de la zone de la barrière.

Résultats

Le jeu offre une meilleure compréhension du caractère probabiliste de la mécanique quantique. Il arrive qu'avec une énergie cinétique supérieure au potentiel du puits, l'électron soit réfléchi, ou qu'il y ait transmission avec une énergie cinétique inférieure au potentiel du puits.

Discussion

Nous aurions pu rendre le jeu plus compatible à l'agrandissement de fenêtre en utilisant le plus souvent possible les tailles horizontales et verticales de l'écran comme objets de référence.
Nous aurions pu rajouter des puits et des barrières non carrées et discrétiser et rendre aléatoire l'énergie donnée par les photons, avec un système de couleur permettant de deviner l'énergie correspondante. Nous aurions pu rendre aléatoire la longueur de la barrière qui actuellement fait 150 pixels, soit 1.50 nm avec notre système.

Pour améliorer le gameplay, nous aurions pu rajouter des champs électromagnétiques et des atomes pour diversifier les manières d'acquérir de l'énergie cinétique à notre electron. Les photons pourraient arriver de directions et d'angles différents.

La comparaison des valeurs de transmissions avec l'applet Java du TP1 de mécanique quantique de L3 de physique de Sorbonne Université (3) nous encourage à penser que nos resultats sont coherents.

Nous sommes limités par le fait que l'ordinateur classique ne simule pas des vrais nombres aléatoires.

Conclusion

Nous sommes parvenus à programmer un jeu simple mettant en avant l'aspect probabiliste de la mécanique quantique lors de la traversée par un électron d'une barrière de potentiel. Ce jeu permet de mieux se rendre compte de l'influence de différents paramètres sur la probabilité de transmission de l'électron.

References

(1) Zachary W. Sherin, Ryan Cheu, Philip Tan, and Gerd Kortemeyer, (2016): **Visualizing Relativity: the OpenRelativity Project**, American Journal of Physics 84, 369-374

(2) Lettre d'Einstein à Born, 1926

(3) TP de physique de L3 de Sorbonne Université

Annexe

Lien Github du projet : https://github.com/TomTomTomahawk/compphysproject

Code :

```
//ligne de compilation : c++ -Wextra -Wall ".cpp" -o ".x" -lsfml-graphics -lsfml-
window -lsfml-system

#include <cstdlib>
#include <iostream>
#include <SFML/Graphics.hpp>
#include <string.h>
#include <cmath>

using namespace std;
using namespace sf;



int main()
{

    double x_electron,y_electron; // position du cercle
    x_electron=400;
    y_electron=300;
    int rayon_electron = 50; // rayon du cercle

    double x_photon,y_photon; // position du cercle
    x_photon=0;
    y_photon=300;
    int rayon_photon = 10; // rayon du cercle

    double Ec=0;
        double Ec_Joules;
    double Ec_max=1;

    double T; // Transmission
    double m=9.109*pow(10,-31);
    double h= 6.62607015*pow(10,-34);
    double hbar=h/(2*M_PI);
        double k1;
        double k2;
        double d;//longueur de la barriere
        double random;//on va comparer T a un nombre entre 0 et 1 pour voir si
transmis
        int pass;//deviendra 1 si passage, 2 si echec

    int windowSizeX = 800, windowSizeY = 600;


    CircleShape photon(rayon_photon);
    photon.setFillColor(Color::Yellow);


    RenderWindow window(VideoMode(windowSizeX, windowSizeY), "A probabilistic
universe");
    window.setFramerateLimit(50);


//Sprites

    Image image;
    image.loadFromFile("electron.png");
    image.createMaskFromColor(Color::Black);
    Texture tex;
    tex.loadFromImage(image);
    Sprite sp;
    sp.setTexture(tex);
    sp.setScale(2*rayon_electron/sp.getLocalBounds().width, 2*rayon_electron/
sp.getLocalBounds().height);
    sp.setOrigin(rayon_electron,rayon_electron);
```

```
//Background

    Image fond;
    fond.loadFromFile("background2.png");

//son affichage
    Texture tex_fond;
    tex_fond.loadFromImage(fond);
    Sprite sp_fond;
    sp_fond.setTexture(tex_fond);


//affichage Energie

    sf::Font font;

    if ( !font.loadFromFile( "./Arial.ttf" ) )
    {
        std::cout << "Error loading file" << std::endl;

        system( "pause" );
    }

    sf::Text text;

    text.setFont( font );

    text.setString( to_string(Ec) );

    text.setCharacterSize( 25 );

    text.setFillColor( sf::Color::White );

    text.setStyle( sf::Text::Style::Bold );

    text.setOutlineColor( sf::Color::Black );
    text.setOutlineThickness( 3 );
    text.setPosition(20,20);

//affichage hauteur du puits

    sf::Font font2;

    if ( !font2.loadFromFile( "./Arial.ttf" ) )
    {
        std::cout << "Error loading file" << std::endl;

        system( "pause" );
    }

    sf::Text text_puits;

    text_puits.setFont( font );

    text_puits.setString( to_string(Ec) );

    text_puits.setCharacterSize( 25 );

    text_puits.setFillColor( sf::Color::White );

    text_puits.setStyle( sf::Text::Style::Bold );

    text_puits.setOutlineColor( sf::Color::Black );
    text_puits.setOutlineThickness( 3 );
    text_puits.setPosition(20,50);
```

```
//affichage Transmission

    sf::Font font3;

    if ( !font3.loadFromFile( "./Arial.ttf" ) )
    {
        std::cout << "Error loading file" << std::endl;

        system( "pause" );
    }

    sf::Text text_trans;

    text_trans.setFont( font );

    text_trans.setString( to_string(Ec) );

    text_trans.setCharacterSize( 25 );

    text_trans.setFillColor( sf::Color::White );

    text_trans.setStyle( sf::Text::Style::Bold );

    text_trans.setOutlineColor( sf::Color::Black );
    text_trans.setOutlineThickness( 3 );
    text_trans.setPosition(20,80);

//affichage statut du jeu, si Game over ou pas, etc.

    sf::Font font4;

    if ( !font4.loadFromFile( "./Arial.ttf" ) )
    {
        std::cout << "Error loading file" << std::endl;

        system( "pause" );
    }

    sf::Text text_status;

    text_status.setFont( font );

    text_status.setCharacterSize( 40 );

    text_status.setFillColor( sf::Color::White );

    text_status.setStyle( sf::Text::Style::Bold );

    text_status.setOutlineColor( sf::Color::Black );
    text_status.setOutlineThickness( 3 );
    text_status.setPosition(windowSizeX/4,windowSizeY/2);


//Affichage puit

    double x1,x2,x3,x4,x5,x6;
    x1=windowSizeX;
    x2=windowSizeX+150;
    x3=windowSizeX+150;
    x4=windowSizeX+300;
    x5=windowSizeX+300;
    x6=windowSizeX+450;

    double U, U_Joules, maxpuits, minpuits, y_puits;
    maxpuits=1;//hauteur maximal du puits
```

```
    minpuits=0;
    y_puits=600;//hauteur minimale, c'est a dire pas trop bas
    U=(maxpuits - minpuits) *(rand()/(RAND_MAX+1.0));//hauteur de puits aleatoire
entre 0 et 1 eV

    VertexArray phaut(LinesStrip,6);
    phaut[0].position = sf::Vector2f(x1, y_puits);
    phaut[1].position = sf::Vector2f(x2, y_puits);
    phaut[2].position = sf::Vector2f(x3, windowSizeY-(U*windowSizeY/maxpuits));//
conversion des eV en pixels
    phaut[3].position = sf::Vector2f(x4, windowSizeY-(U*windowSizeY/maxpuits));
    phaut[4].position = sf::Vector2f(x5, y_puits);
    phaut[5].position = sf::Vector2f(x6, y_puits);

    // on fait tourner le programme tant que la fenêtre n'a pas été fermée

    while (window.isOpen()) {
        // on traite tous les évènements de la fenêtre qui ont été générés depuis
la dernière itération d
        Event event;
        while (window.pollEvent(event)) {
            // fermeture de la fenêtre lorsque l'utilisateur le souhaite
            if (event.type == Event::Closed)
            window.close();
        }
        window.clear(Color::Black);

if (x_electron >=x1 && x_electron <=x5 && pass!=2){
    if (x_electron >= x1 && x_electron <= x4 - 2*rayon_electron) {//jeu en periode
de puits

text_status.setString( "God is playing dice..." );
        if (y_electron+rayon_electron < windowSizeY-(Ec*windowSizeY)/Ec_max){

        y_electron = y_electron +4;

        }

        if (y_electron + rayon_electron > windowSizeY-(Ec*windowSizeY)/Ec_max){

        y_electron = y_electron -4;

        }

        Ec_Joules=Ec*1.60218*pow(10,-19);
        U_Joules=U*1.60218*pow(10,-19);
        k1= sqrt((2*m*Ec_Joules)/(hbar*hbar));
        k2=sqrt((2*m*(Ec_Joules-U_Joules))/(hbar*hbar));
        d=(x5-x2)*pow(10,-11);//Conversion des pixels en distance. 1 pixel =
10^-11 metre

    if(Ec_Joules > U_Joules){
        T=(4.0*k1*k1*k2*k2)/(4.0*k1*k1*k2*k2+(k1*k1-k2*k2)*(k1*k1-
k2*k2)*sin(k2*d)*sin(k2*d)); // Transmission pour E>V0
    } else {

        T=1/(1+U_Joules*U_Joules/(4*Ec_Joules*(U_Joules-
Ec_Joules))*pow(sinh(sqrt(2*m*(U_Joules-Ec_Joules))/(hbar)*d),2)); //Transmission
pour E<V0

    }
random = rand()/(RAND_MAX+1.0);//on genere un nombre aleatoire que l'on va ensuite
comparer a T

    }
```

```
        if (x_electron >= x4 - 2*rayon_electron && x_electron <= x5) {//jeu en
periode de puits


                if (T>random){
                        text_status.setString( "Carry on" );
                        pass=1;

                } else {

        pass=2;


                }

        }
}

if(pass==2){
        x_electron=x_electron-2;
        text_status.setString( "Not this time, sorry mate" );

        if (x_electron <-200){
                T=0;
                text_status.setString( "Press Space to play again" );
        }
}


if (sf::Keyboard::isKeyPressed(sf::Keyboard::Space)) {//restart le jeu avec un
nouveau puits
                x_electron=400;
                y_electron=300;
                Ec=0;
                pass=0;

                x6=0;
        text_status.setString( "" );

        }



    if ((x_electron <=x1 || x_electron >=x5) && pass!=2) {//jeu en periode
normale, sans puits
text_status.setString( "" );//on clear le texte

        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up) && y_electron > 0.0) {
            y_electron = y_electron -10;
            // move up...
        }


        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down) && y_electron <
windowSizeY-100.0) {
            y_electron = y_electron +10;
            // move down...
        }



    if (x_photon >= windowSizeX) {//hauteur de lancement du photon et sa
reinitialisation lorsqu'il depasse la fenetre
        x_photon=0;
        y_photon=600.0*(rand()/(RAND_MAX+1.0));
```

```
    }

    if (abs((x_photon+rayon_photon) - (x_electron+rayon_electron))<60 &&
abs((y_photon+rayon_photon) - (y_electron+rayon_electron))<60){

        if (Ec<Ec_max-0.008){

        Ec=Ec+0.008;
        } else {
        Ec=Ec_max;
        }
        x_photon=x_photon+3;
    } else {
        if (Ec>0.0008){
        Ec = Ec - 0.0008;
        } else {
        Ec=0;
        }
        x_photon=x_photon+10;
    }

}//fin scenario de jeu normal

    if (x6 <= 0) {//gerer la position du puits, 500 equivaut à 500 frames, soit 10
secondes car 50 f/s
    x1=windowSizeX;
    x2=windowSizeX+150;
    x3=windowSizeX+150;
    x4=windowSizeX+300;
    x5=windowSizeX+300;
    x6=windowSizeX+450;
    U=(maxpuits - minpuits) *(rand()/(RAND_MAX+1.0));
    }

    x1=x1-1;
    x2=x2-1;
    x3=x3-1;
    x4=x4-1;
    x5=x5-1;
    x6=x6-1;

    phaut[0].position = sf::Vector2f(x1, y_puits);
    phaut[1].position = sf::Vector2f(x2, y_puits);
    phaut[2].position = sf::Vector2f(x3, windowSizeY-(U*windowSizeY/maxpuits));
    phaut[3].position = sf::Vector2f(x4, windowSizeY-(U*windowSizeY/maxpuits));
    phaut[4].position = sf::Vector2f(x5, y_puits);
    phaut[5].position = sf::Vector2f(x6, y_puits);




    window.draw(sp_fond);

    text.setString( "Electron's kinetic energy = " + to_string(Ec) + " eV");
    window.draw( text );

    text_trans.setString( "Transmission probability = " + to_string(T*100.0) + "
%");
    window.draw( text_trans );

    text_puits.setString( "Barrier's height = " + to_string(U) + " eV");
    window.draw( text_puits );

    sp.setPosition(x_electron+5,y_electron+5);
    window.draw(sp);
```

```
    photon.setPosition(x_photon,y_photon);
    window.draw(photon);

    window.draw(phaut);
window.draw( text_status );

    window.display();
    }//fin boucle while SFML


return 0;

}
```