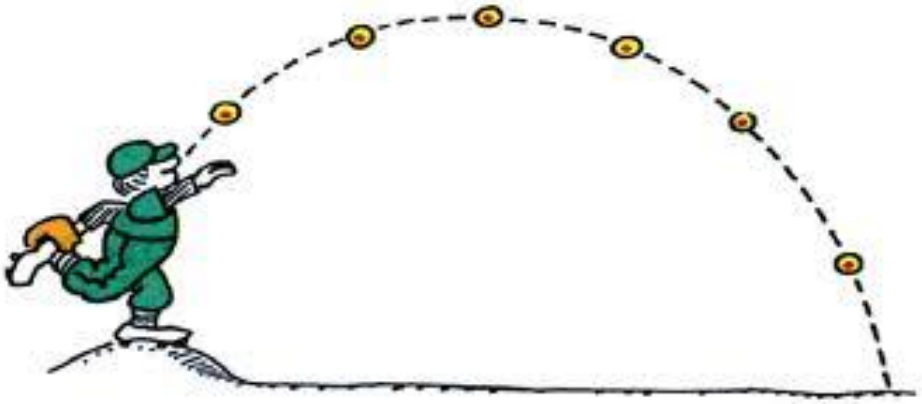
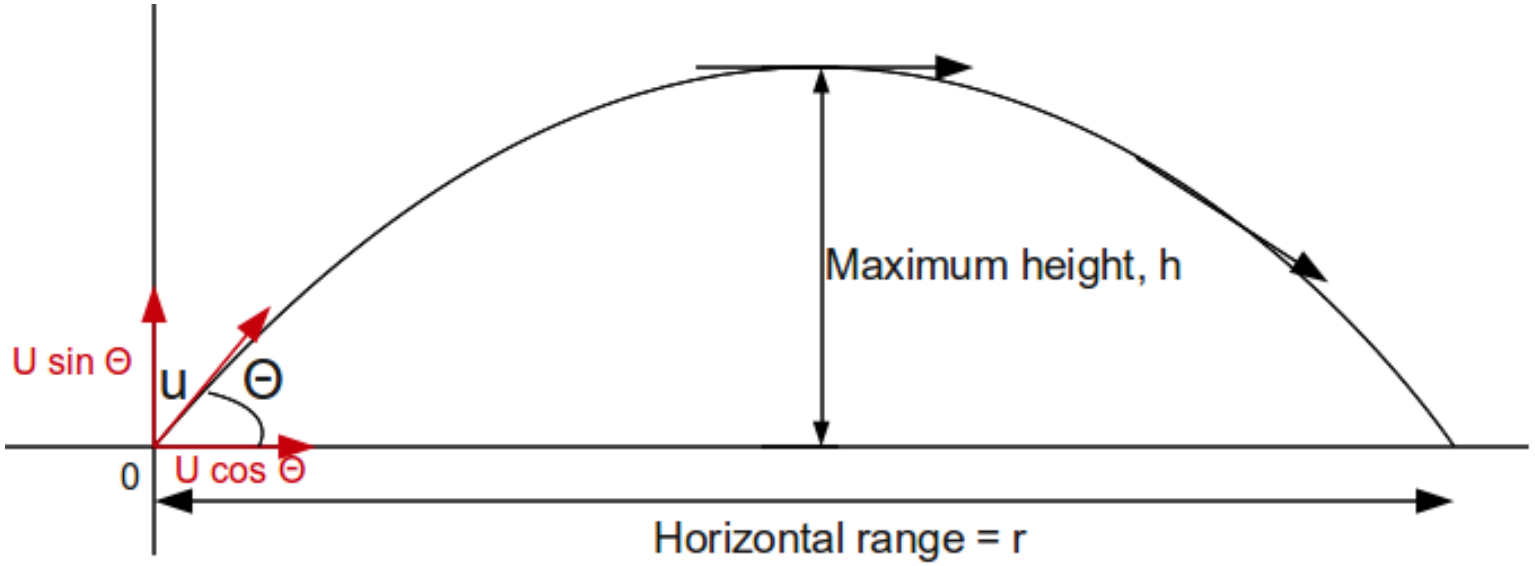
Implementing Mathematical Models

Mohammad Imrul Jubair

Lets implement projectile motion



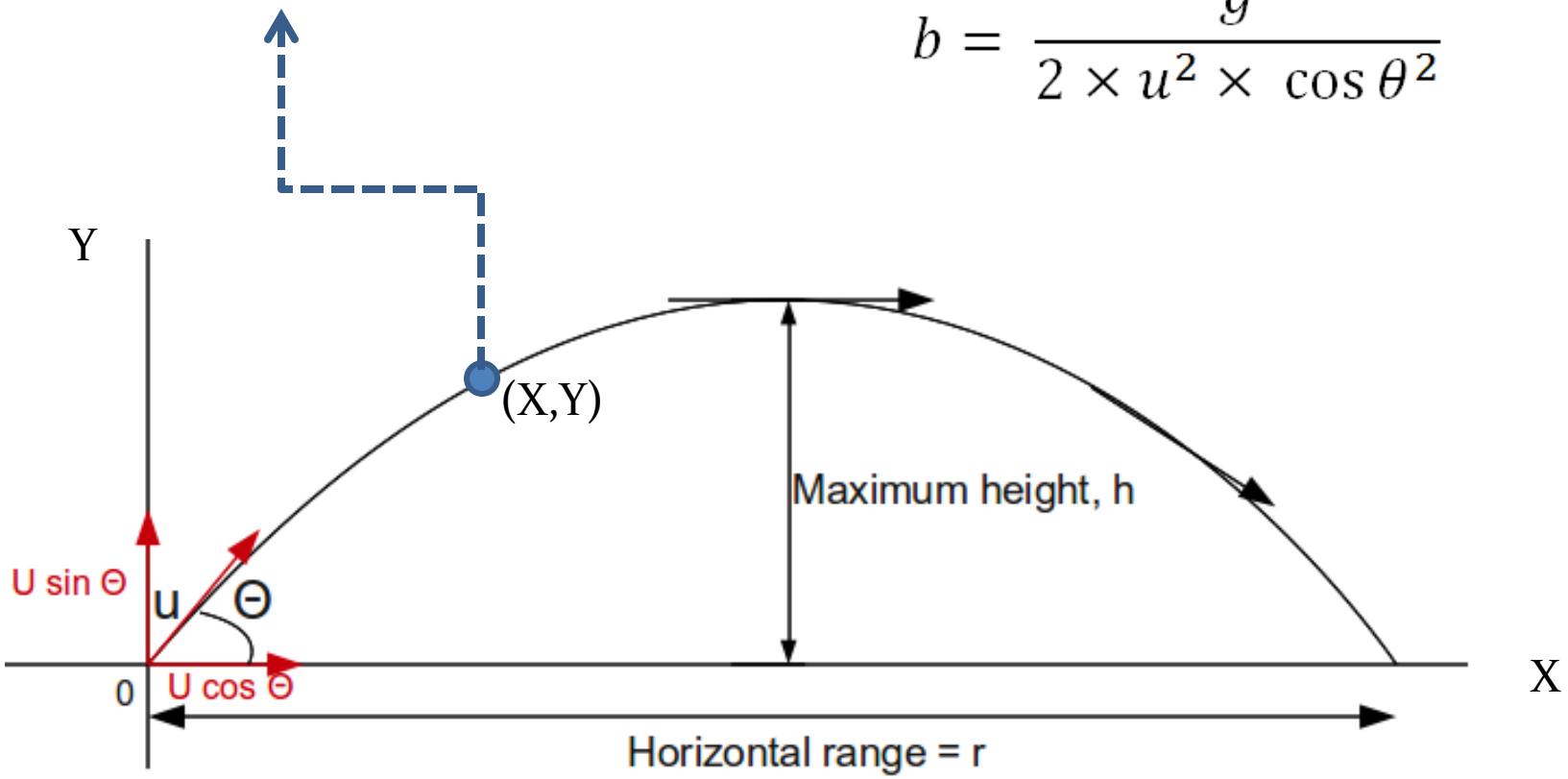


Lets implement projectile motion

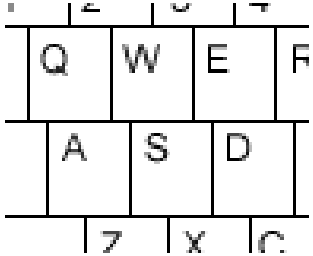
$$y = ax - bx^2$$

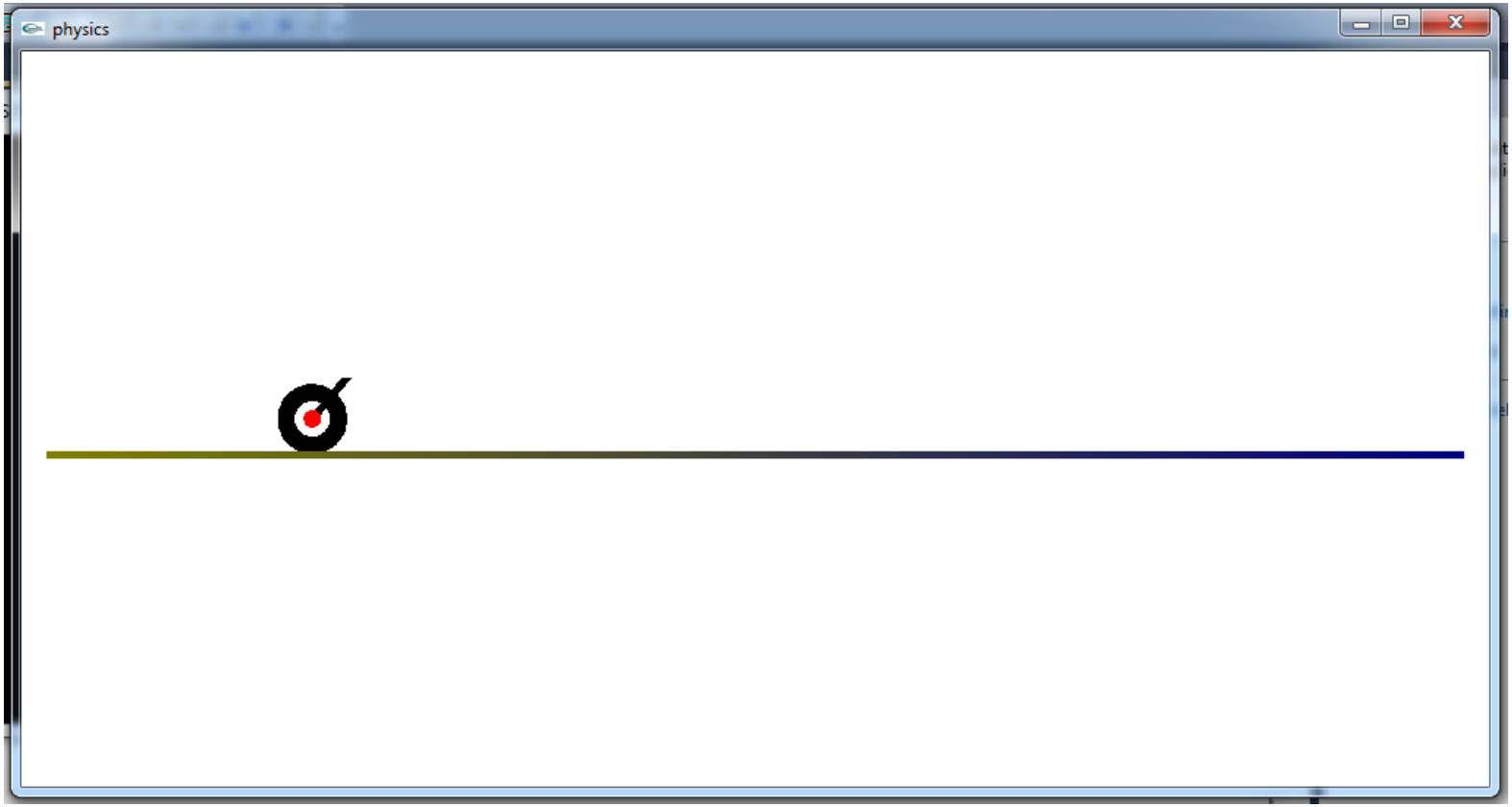
$$a = \tan(\theta)$$

$$b = \frac{g}{2 \times u^2 \times \cos\theta^2}$$

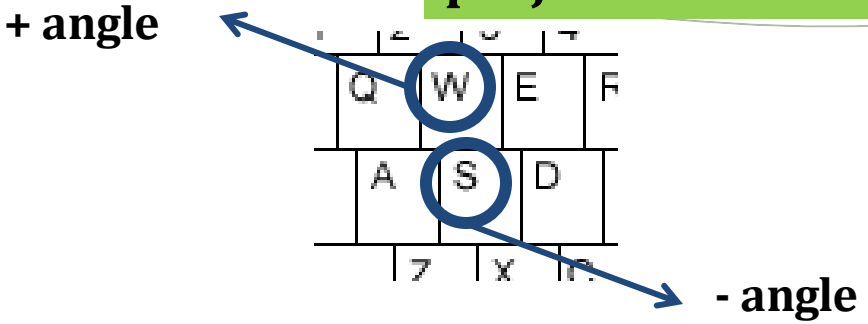


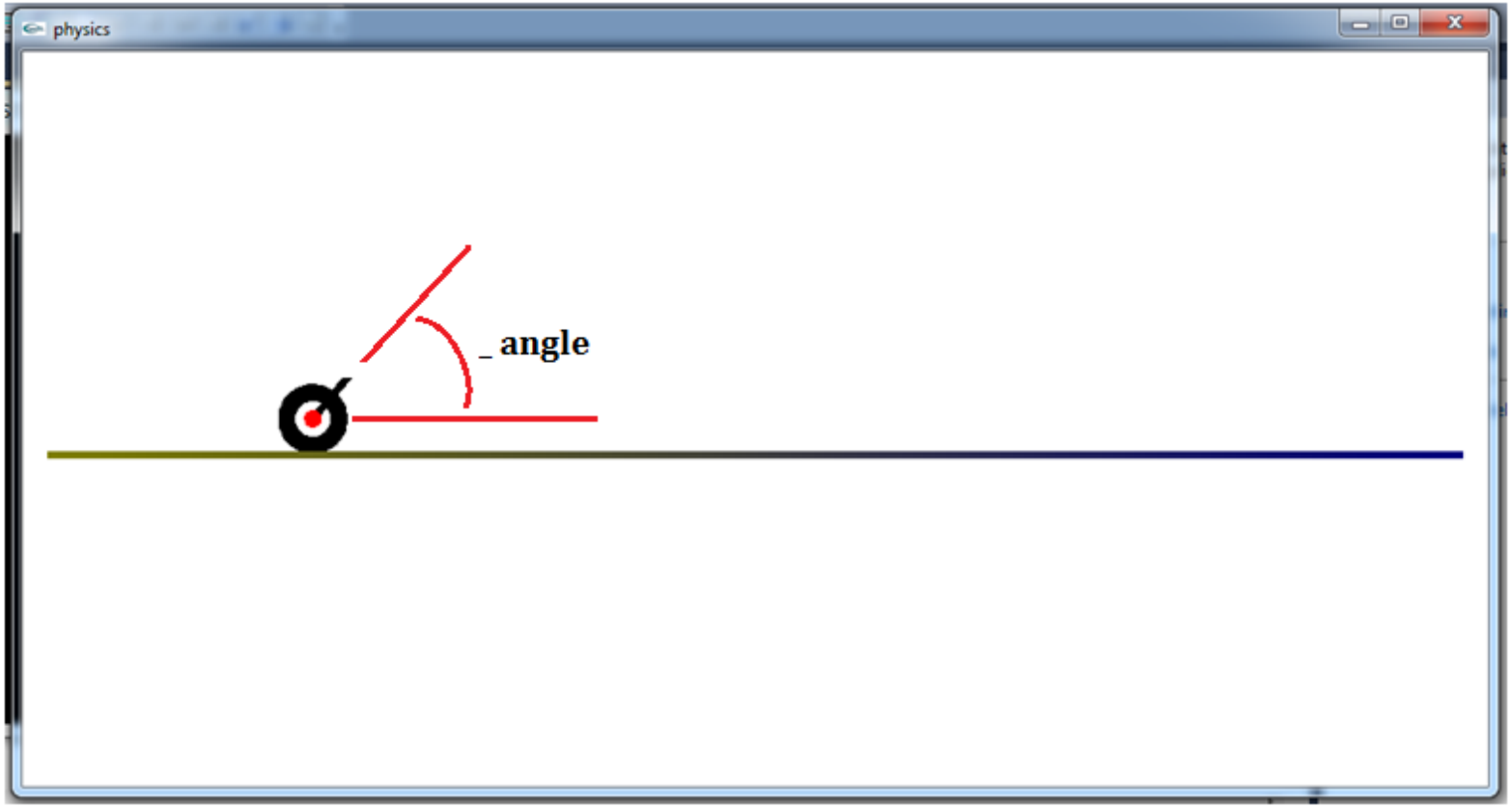
projectile motion in openGL



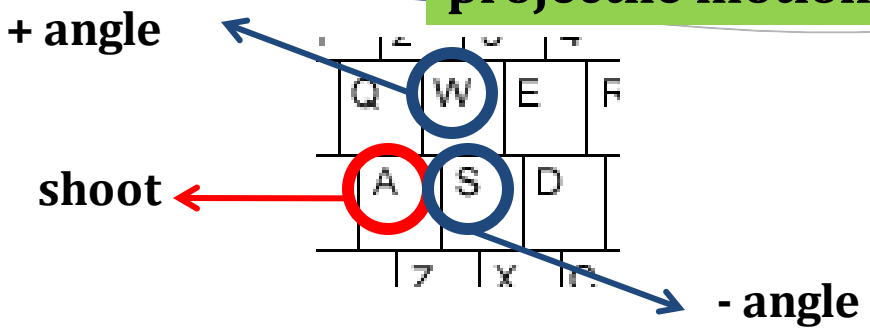


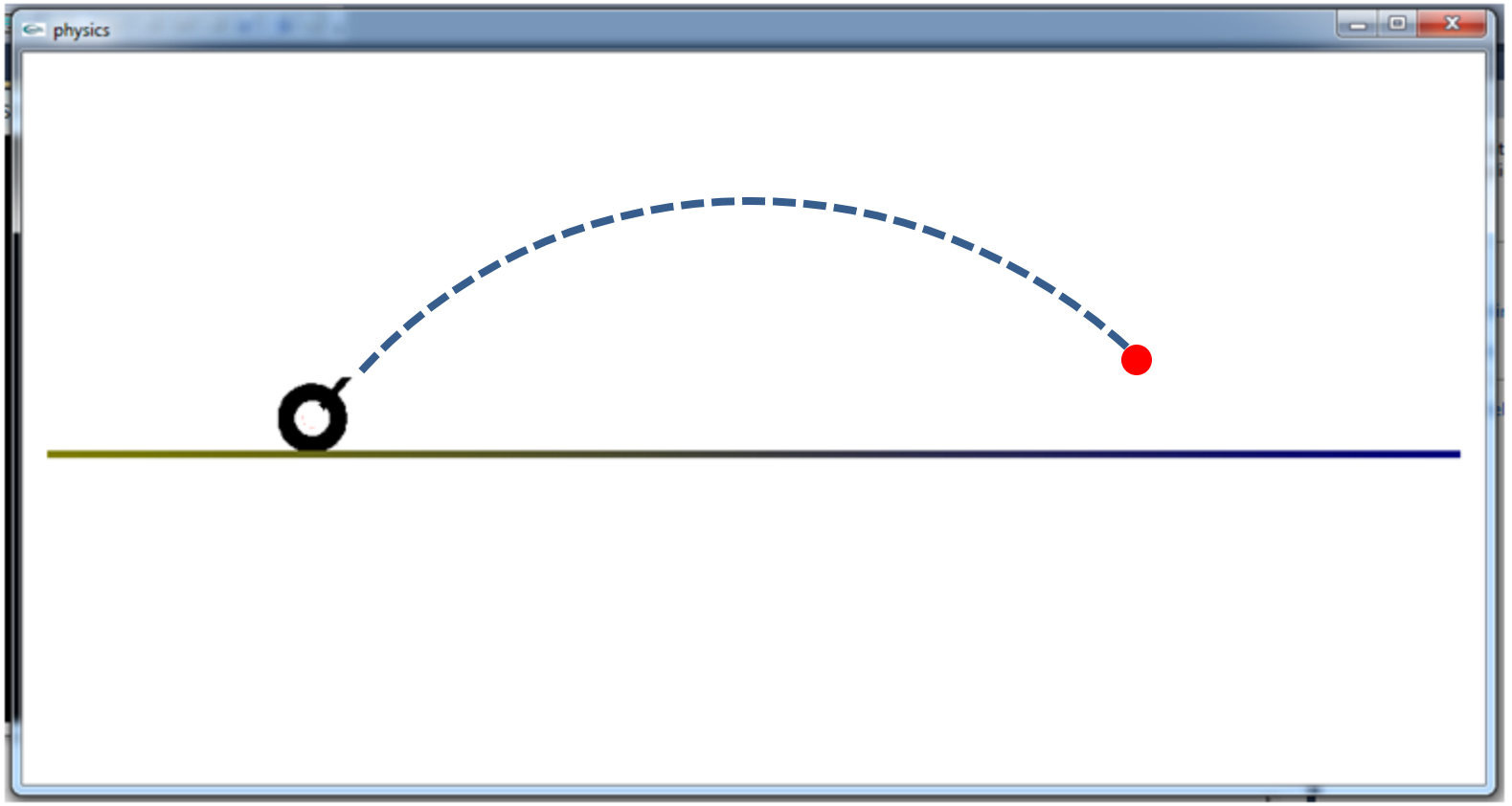
projectile motion in openGL





projectile motion in openGL

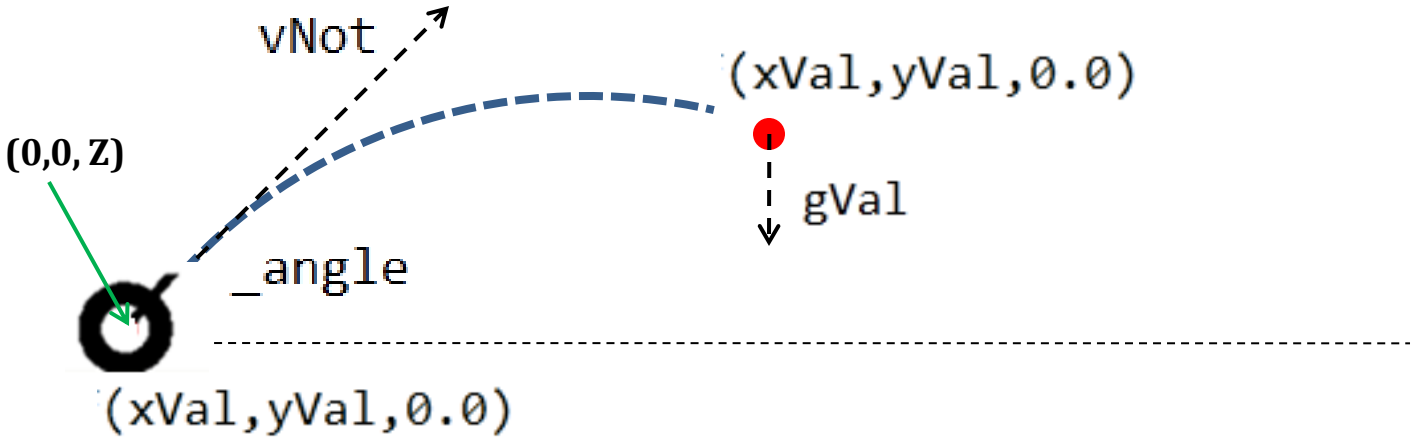




Lets Code

Initializations →

```
float _angle = 0.0;
float gVal = 9.8;
float vNot = 10.0;
float xVal = 0.0, yVal = 0.0;

float a = 0.0;
float b = 0.0;
```



.... Functions to draw

```
void Draw() { ... }
void theGun() { ... }
void theSurface() { ... }
void theBullet() { ... }
```

.... The bullet will be moving

```
void Draw() { ... }
void theGun() { ... }
void theSurface() { ... }
void theBullet() { ... }
```

```
glPushMatrix();
        glRotatef(_angle,0.0,0.0,1.0);
        theGun();
glPopMatrix();
```

.... The bullet will be moving

```
void Draw() { ... }
void theGun() { ... }
void theSurface() { ... }
void theBullet() { ... }
```

```
glPushMatrix();
        glRotatef(_angle,0.0,0.0,1.0);
        theGun();
glPopMatrix();
```

```
void theBullet()
{   glColor3f(1.0, 0.0, 0.0);
    glTranslatef(xVal,yVal,0.0);
    glutSolidSphere(0.1, 15.0, 2.0);   }
```

..... Key press handling

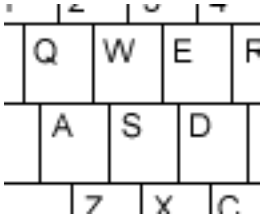
```
void Draw() { ... }
void theGun() { ... }
void theSurface() { ... }
void theBullet() { ... }
```

```
void handleKeypress(unsigned char key, int x, int y){
    switch (key) {
    case 'w':
        _angle += 1.0;
        glutPostRedisplay();
        break;
    case 's':
        _angle -= 1.0;
        glutPostRedisplay();
        break;
    case 'a':
        update(0);
        break;}
}
```

..... Key press handling

```
void Draw() { ... }
void theGun() { ... }
void theSurface() { ... }
void theBullet() { ... }
```

```
void handleKeypress(unsigned char key, int x, int y){
    switch (key) {
    case 'w':
        _angle += 1.0;
        glutPostRedisplay();
        break;
    case 's':
        _angle -= 1.0;
        glutPostRedisplay();
        break;
    case 'a':
        update(0);
        break;}
}
```



Updating the (X,Y)

```
void handleKeypress(unsigned char key,
    switch (key) {
    case 'w':
        _angle += 1.0;
        glutPostRedisplay();
        break;
    case 's':
        _angle -= 1.0;
        glutPostRedisplay();
        break;
    case 'a':
        update(0);
        break;}
}
```

```
void update(int value)
{
    theCalculate();
    xVal += 0.003;
    yVal = a*xVal - b*xVal*xVal;
    glutPostRedisplay();
    if (yVal > -0.8 && xVal < 12.0)
        glutTimerFunc(1.0,update,0);
    else
    {   xVal = 0.0;
        yVal = 0.0;  }
}
```

Updating the (X,Y) : The equation

```
float a = 0.0;
float b = 0.0;
void theCalculate()
{   a = tan(_angle*(3.1416/180.0));
    b = gVal/(2*vNot*vNot*cos(_angle*(3.1416/180.0))*cos(_angle*(3.1416/180.0))); }
```

```
void handleKeypress(unsigned char key,
    switch (key) {
    case 'w':
        _angle += 1.0;
        glutPostRedisplay();
        break;
    case 's':
        _angle -= 1.0;
        glutPostRedisplay();
        break;
    case 'a':
        update(0);
        break;}
}
```

```
void update(int value)
{
    theCalculate();
    xVal += 0.003;
    yVal = a*xVal - b*xVal*xVal;
    glutPostRedisplay();
    if (yVal > -0.8 && xVal < 12.0)
        glutTimerFunc(1.0,update,0);
    else
    {   xVal = 0.0;
        yVal = 0.0;  }
}
```

Updating the (X,Y) : The equation

```
float a = 0.0;
float b = 0.0;
void theCalculate()
{   a = tan(_angle*(3.1416/180.0));
    b = gVal/(2*vNot*vNot*cos(_angle*(3.1416/180.0))*cos(_angle*(3.1416/180.0))); }
```

$$a = \tan(\theta)$$

$$b = \frac{g}{2 \times u^2 \times \cos\theta^2}$$

```
void update(int value)
{
    theCalculate();
    xVal += 0.003;
    yVal = a*xVal - b*xVal*xVal;
    glutPostRedisplay();
    if (yVal > -0.8 && xVal < 12.0)
        glutTimerFunc(1.0,update,0);
    else
    {   xVal = 0.0;
        yVal = 0.0;  }
}
```

Updating the (X,Y) : The equation

```
float a = 0.0;
float b = 0.0;
void theCalculate()
{   a = tan(_angle*(3.1416/180.0));
    b = gVal/(2*vNot*vNot*cos(_angle*(3.1416/180.0))*cos(_angle*(3.1416/180.0))); }
```

$$a = \tan(\theta)$$

$$b = \frac{g}{2 \times u^2 \times \cos\theta^2}$$

$$y = ax - bx^2$$

```
void update(int value)
{
    theCalculate();
    xVal += 0.003;
    yVal = a*xVal - b*xVal*xVal;
    glutPostRedisplay();
    if (yVal > -0.8 && xVal < 12.0)
        glutTimerFunc(1.0,update,0);
    else
    {   xVal = 0.0;
        yVal = 0.0;  }
}
```

Updating the (X,Y) : The equation

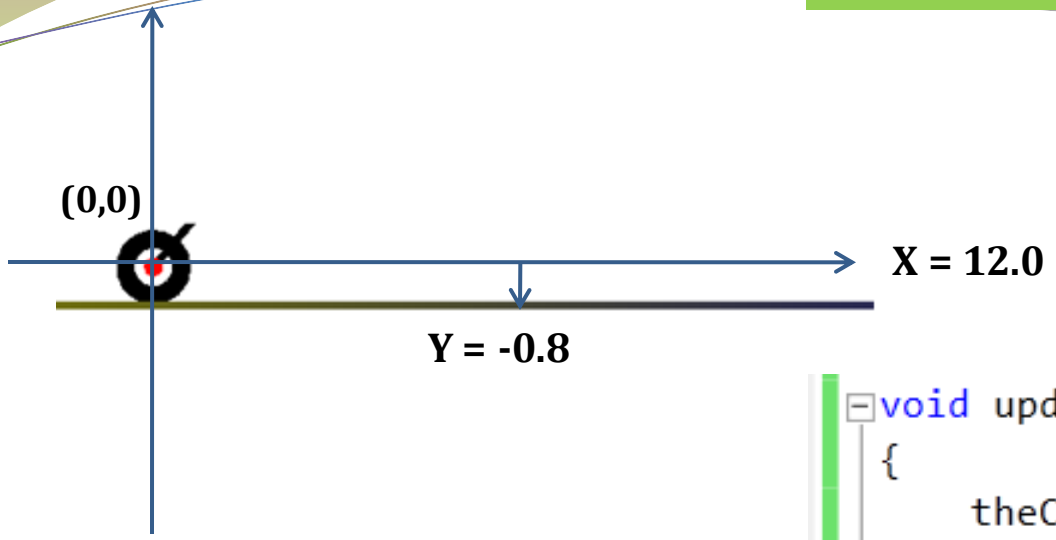
```
float a = 0.0;
float b = 0.0;

void theCalculate()
{   a = tan(_angle*(3.1
    b = gVal/(2*vNot*vN
```

```
void theBullet()
{   glColor3f(1.0, 0.0, 0.0);
    glTranslatef(xVal,yVal,0.0);
    glutSolidSphere(0.1, 15.0, 2.0);  }
```

```
void update(int value)
{
    theCalculate();
    xVal += 0.003;
    yVal = a*xVal - b*xVal*xVal;
    glutPostRedisplay();
    if (yVal > -0.8 && xVal < 12.0)
        glutTimerFunc(1.0,update,0);
    else
    {   xVal = 0.0;
        yVal = 0.0;  }
}
```

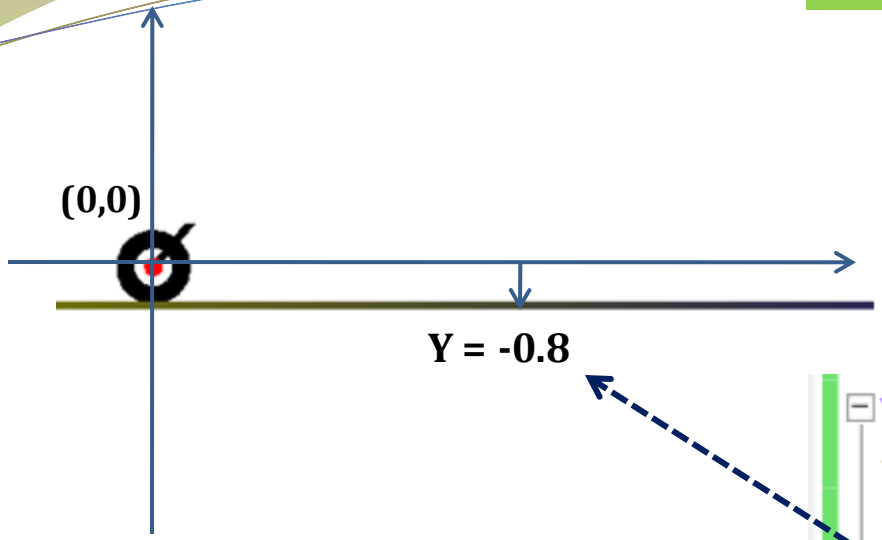
$$y = ax - bx^2$$

Limiting the scenario.....



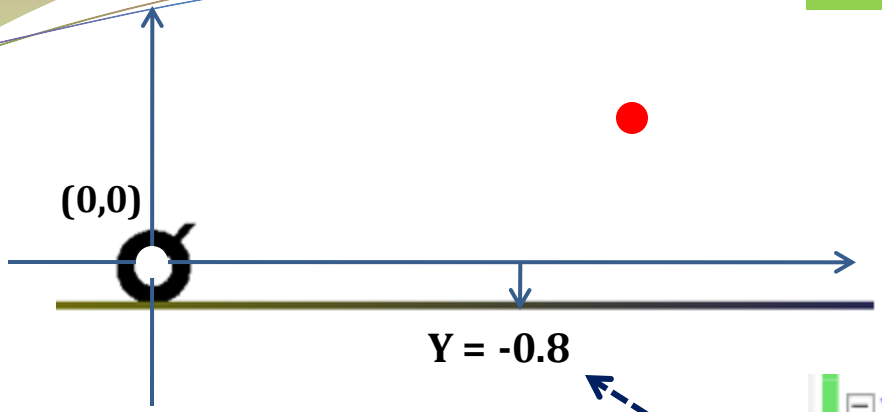

```
void update(int value)
{
    theCalculate();
    xVal += 0.003;
    yVal = a*xVal - b*xVal*xVal;
    glutPostRedisplay();
    if (yVal > -0.8 && xVal < 12.0)
        glutTimerFunc(1.0,update,0);
    else
    {   xVal = 0.0;
        yVal = 0.0;  }
}
```

projectile motion is ready




```
void update(int value)
{
    theCalculate();
    xVal += 0.003;
    yVal = a*xVal - b*xVal*xVal;
    glutPostRedisplay();
    if (yVal > -0.8 && xVal < 12.0)
        glutTimerFunc(1.0,update,0);
    else
    {   xVal = 0.0;
        yVal = 0.0;  }
}
```

projectile motion is ready




```
void update(int value)
{
    theCalculate();
    xVal += 0.003;
    yVal = a*xVal - b*xVal*xVal;
    glutPostRedisplay();
    if (yVal > -0.8 && xVal < 12.0)
        glutTimerFunc(1.0,update,0);
    else
    {   xVal = 0.0;
        yVal = 0.0;  }
}
```