Simple Quick Dirty
Collision Detection
For Beginners

Contents

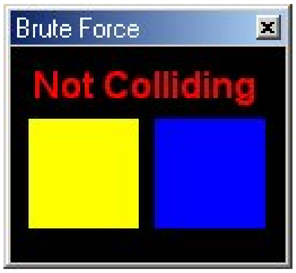
2D Collision Detection

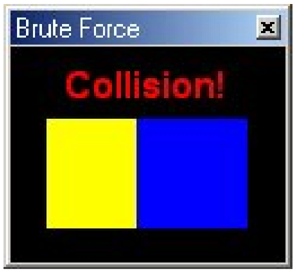
2D Collision Detection Methods

There are many types of 2D-collision detection that can be done. Such methods include brute force, bit array, bounding sphere, bounding box, axis sorting, tile based, sprite bounds, grid, and static. As you can see there are many different types of 2D-collision detection “styles,” but for the sake of this article we will deal with brute force and bounding sphere. Brute force, bit array, and tile based are usually the most common. Lets deal with brute force first. The Bounding Box and Axis Sorting is almost identical to the brute force.

Brute Force Collision Detection

In Visual Basic a brute force algorithm is as simple as the following. This algorithm is self-explanatory, it simply just checks the bounds of both Sprites.
Note: For simplicity, we will use picture box’s.





Function:

```
 Public Function Check_Collision_BF(Sprite1 As PictureBox, Sprite2 As PictureBox) As
Boolean
   If Sprite1.Left > Sprite2.Left + Sprite2.Width Or _
       Sprite1.Left + Sprite1.Width < Sprite2.Left Or _
       Sprite1.Top > Sprite2.Top + Sprite2.Height Or _
       Sprite1.Top + Sprite1.Height < Sprite2.Top Then
     Check_Collision_BF = False
   Else
     Check_Collision_BF = True
   End If

 End Function
```

How to Use:

```
  If Check_Collision_BF(pic2DColl(0), pic2DColl(1)) = True Then
    Label1.Caption = "Collision!"
  Else
    Label1.Caption = "Not Colliding"
  End If
```

Back to Top

Bounding Sphere Collision Detection

My personal opinion (*if it accounts for anything*) is that bounding spheres is far more easily to compute than anything. Its so simple its child’s play, but people hear “bounding volumes” or “bounding sphere” and read a text book and it says something like “$(X^2 + X^2) – (Y^2 + Y^2)$” and freak out. It’s nothing like that. There is actually two ways of doing this and I’ll show you both. One way is to figure (Sprite1.Pos = Sprite2.Pos – Radius) or use the positions and two radii. Sounds complicated, but like I said, it is simple. Here is example one.

Function 1 – Position, Position, Radii:

```
Public Function SphereCollision_2D1(X1 As Single, Y1 As Single, X2 As Single, Y2 As Single,
Radius As Single) As Boolean

  If X1 > X2 - Radius And X1 < X2 + Radius And _
    Y1 > Y2 - Radius And Y1 < Y2 + Radius Then
    SphereCollision_2D1 = True
  Else
    SphereCollision_2D1 = False
  End If

 End Function
```

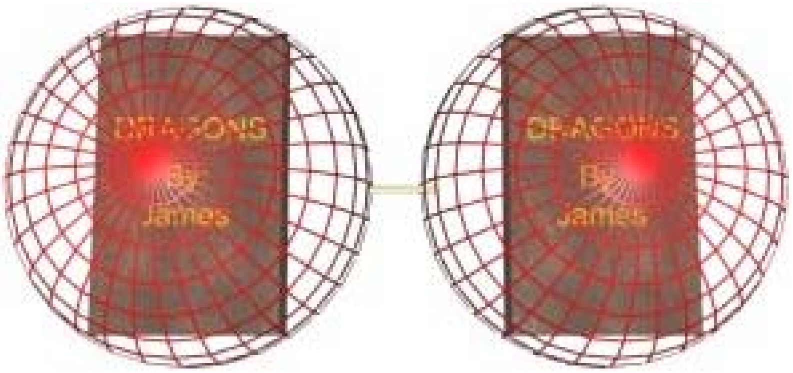
How to Use:

```
  If SphereCollision_2D1 (pic2DColl(0).Left, pic2DColl(0).Top, pic2DColl(1).Left,
pic2DColl(1).Top, 3) = True Then
   Label1.Caption = "Collision!"
 Else
   Label1.Caption = "Not Colliding"
 End If
```

Back to Top

Function 2 – Position, Position, Radii, Radii:

Function 2 is pretty much does the same thing but it uses actual Distance to calculate the outcome. It takes the radius of both sprites then tests the distance between the “spheres”(radii).



```
Public Function SphereCollision_2D2(X1 As Single, Y1 As Single, X2 As Single, Y2 As Single,
Radius1 As Single, Radius2 As Single) As Boolean

   Dim Dist As Single

   Dist = Radius1 + Radius2

   If  ((Sqr(((X1 – X2) * (X1 – X2)) + _
              ((Y1 – Y2) * (Y1 – Y2)))) < Dist) Then
     SphereCollision_2D2 = True
   Else
     SphereCollision_2D2 = False
   End If

  End Function
```

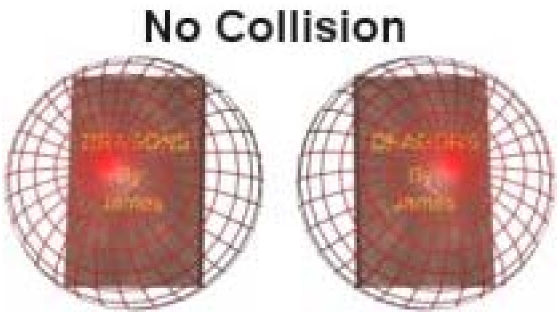
How to Use:

```
   If SphereCollision_2D2 (pic2DColl(0).Left, pic2DColl(0).Top, pic2DColl(1).Left,
pic2DColl(1).Top, 3, 3) = True Then
    Label1.Caption = "Collision!"
  Else
    Label1.Caption = "Not Colliding"
  End If
```

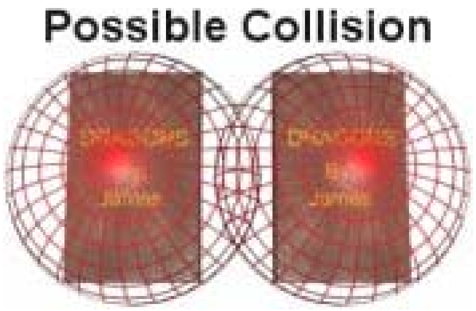
Back to Top

Uses For – Position, Position, Radii, Radii

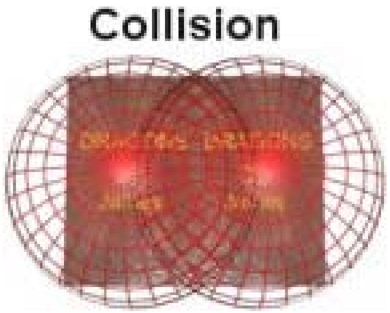
This method is good to detect collisions more accurately. Method one is decent but it can miss detections. If you want accuracy use method two with the following explanation.



Here we can see clearly that there is no collision what so ever. Therefore, there is no need to check for collisions.



Here the two bounding spheres are colliding, so we start checking for collisions between the books.



Here the two books are definitely colliding.

This method saves a lot of processing power because you're not checking for collisions at each frame, only the bounding spheres which do not take that much to compute. So you have two choices, the one described that checks collision with the books or you can simply check collisions with the spheres.

Back to Top

So What About 3D Collision Detection?

3D-collision detection works the same as the two 2D bounding sphere examples I explained. All you need to do is add the (z) coordinates, not to hard.

Look at the 2D Demo and look at the comment. I added 3D Collision Detection to give you an example.

Back to Top

This article is for education purposes only. There is not warranty what so ever. There is no implied warranty. I am not responsible for any damages this software may cause to your computer. Use at your own risk.

This article and source code is provided to the best of my knowledge. If there are any errors and/or misinformation please e-mail me at james-d-21@bright.net **so I can fix them.**

Thank you,
-James-

Back to Top