ゲーム制作関連

C++とゲーム制作に関する話題

- カウントダウンタイマークラスを作ってみよう z
- AABB、OBBとは
- 球と球の当たり判定(3D)（by チョコミント）
- カプセルコライダーについて（by チョコミント）
- 箱(AABB)と球の当たり判定
- 2点間の距離

カウントダウンタイマークラスを作ってみようｚ

CDTimer.h

```
#pragma once
#include "Engine/GameObject.h"

class CDTimer :
    public GameObject
{
private:
 double CountDownTimer_;  //現在の残り時間
 double TIMER_INIT_TIME_; //初期時間
 bool isTimerRun_;//タイマーが動いているかどうか？
 DWORD oldTime_;
public:
 //コンストラクタ
 //引数：parent  親オブジェクト（SceneManager）
 CDTimer(GameObject *parent);
 CDTimer(GameObject *parent, double _initTime);


 //初期化
 void Initialize() override;

 //更新
 void Update() override;

 //描画
 void Draw() override;

 //開放
 void Release() override;

public:

 bool IsTimeOver();//タイマーは０になりましたか？ＹＥＳ？ＮＯ？
 void ResetTimer();//タイマーを初期時間に戻す
 void StartTimer();//タイマーをスタートします
 void StopTimer();//タイマーをストップします
 void SetInitTime(double cdTime) { TIMER_INIT_TIME_ = cdTime; ResetTimer(); }
 double GetTime() { return(CountDownTimer_); }
};
```

CDTimer.cpp

```
#include "CDTimer.h"


const int DEF_TIMER_TIME{ 5 };

CDTimer::CDTimer(GameObject* parent)
 :TIMER_INIT_TIME_(DEF_TIMER_TIME),
 CountDownTimer_(DEF_TIMER_TIME),
 isTimerRun_(true)
{
 oldTime_ = timeGetTime();
};
```

```
CDTimer::CDTimer(GameObject* parent, double _initTime)
 :TIMER_INIT_TIME_(_initTime),
 CountDownTimer_(_initTime),
 isTimerRun_(true)
{
 oldTime_ = timeGetTime();
};

bool CDTimer::IsTimeOver()
{
 return(CountDownTimer_ <= 0);
}

void CDTimer::ResetTimer()
{
 CountDownTimer_ = TIMER_INIT_TIME_;
 StartTimer();
}

void CDTimer::StartTimer()
{
 isTimerRun_ = true;
}

void CDTimer::StopTimer()
{
 isTimerRun_ = false;
}

void CDTimer::Initialize()
{

}

void CDTimer::Update()
{
 DWORD nowTime = timeGetTime();
 //下staticは要らんかった。。。
 //static int deltaTime = nowTime - oldTime_;
 int deltaTime = nowTime - oldTime_;
 if (isTimerRun_)
  CountDownTimer_ = CountDownTimer_ - (float)deltaTime/1000.0;
 oldTime_ = nowTime;
}

void CDTimer::Draw()
{
}

void CDTimer::Release()
{
}
```

AABB、OBBとは

AABB、OBBとは

こんにちは、チョコミントです。
今回はAABBとOBBについて聞きなれてない人も多いと思うので簡単に説明します。

- AABBとは**軸平行境界ボックス**といって**箱の各面の法線が座標軸と平行**なものです。
  箱の横と縦と奥へのベクトルそれぞれがXYZと平行ということですね。
- OBBとは**有向境界ボックス**といって**箱の各面の法線が座標軸と平行ではない**も のです。
  なんとなく理解できたでしょうか。

AABBとOBBでは当たり判定をする際に処理速度が変わる！？

AABBとOBBでは**当たり判定をする際の処理速度**がかなり変わってきます。
箱の各軸が**座標軸と平行ではない**だけで衝突判定がものすごくめんどくさいんです.....
簡単にいうと**箱の各軸を分離軸としてその分離軸で射影してあげる**必要があるからです。
そして射影しているときに**分離超平面がみつかれば衝突していない**ということになります。

この手法は**様々な3Dプリミティブの当たり判定**で使われます。

1. OBBとOBB
2. OBBと球
3. OBBとカプセル
4. OBBと三角形

これ以外でもありそうですが.....
文章だけ読むと複雑そうには思いますが、実際に実装してみると仕組みさえ理解してしまえばわかると思います。
分離軸で射影して分離超平面を見つける手法もいつか記事で書きたいと思います。

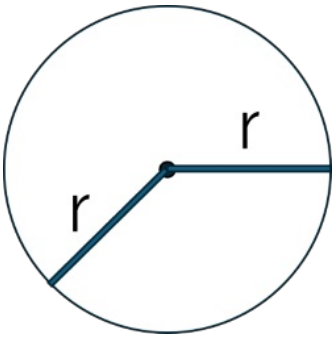
最後に

今回はAABBとOBBについての記事でした。
分からないことや質問などがあれば気軽にコメント待っています！！

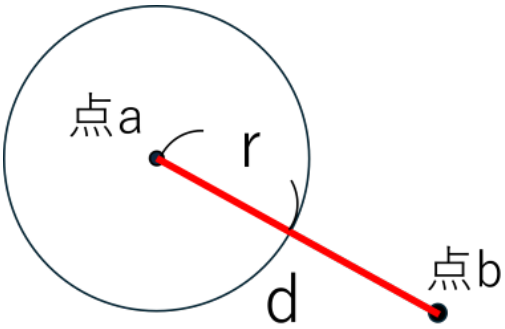
球と球の当たり判定(3D)（by チョコミント）

球と球の当たり判定

こんにちは、チョコミントです。 今回から当たり判定について軽く触れていこうかなと思います。 まずは比較的簡単な**球と球の当たり判定**についてです。 もう知っている方も多いと思いますが、よければ見ていってください。



球は**中点から円周の距離が全方向半径(r)の長さ**で構成されています。 では、球と点が衝突しているか確かめるにはどのように考えればいいのでしょうか。



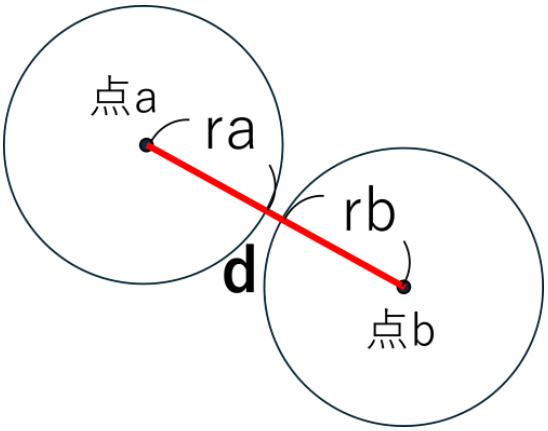
上の図では点a(球体に中点)から点bの距離をdとし、球体の半径をrと定義しています。 もうぴんっときた人もいるかもしません。 **dの距離がr(半径)以下だったら衝突**しているのです。 ってことは半径はもうわかっているので、**点aと点bの距離**が分かれば衝突しているかわかりそうですね！！

点と点の距離

点と点の距離は**三平方の定理**を利用して求めることができます。 3次元の場合はこのような感じです。

```
XMFLOAT3 c = { a.x - b.x,a.y - b.y,a.z - b.z };
return sqrtf((c.x * c.x) + (c.y * c.y) + (c.z * c.z));
```

簡単に説明するとX,Y,Z軸でそれぞれの始点(例:点a)から終点(例:点b)へのベクトルを求めて、 それらの合成ベクトルの長さが距離となるのです。 点と点の距離が分かってしまえばもう球と球の当たり判定は簡単です。



この図をみて理解した人も多いと思いますが、 **dの距離が(ra(半径) + rb(半径))以下だったら衝突**しているのです。

```
//点と点の距離を求める
XMFLOAT3 a, XMFLOAT3 b;
XMFLOAT3 c = { a.x - b.x,a.y - b.y,a.z - b.z };
float d = sqrtf((c.x * c.x) + (c.y * c.y) + (c.z * c.z));

float ra = 0.5f; //球体aの半径
float rb = 1.0f; //球体bの半径

//aとbの距離がそれぞれの半径を足した合計値以下なら当たっている
if (ra + rb >= d)
{
 //衝突している
}
```

サンプルプログラムです。

最後に

今回は球と球の当たり判定についての記事でした。 物足りない人も多かったのではないでしょうか。 次回は球と箱(AABB)の当たり判定について記事を書いてみます。

カプセルコライダーについて（by チョコミント）

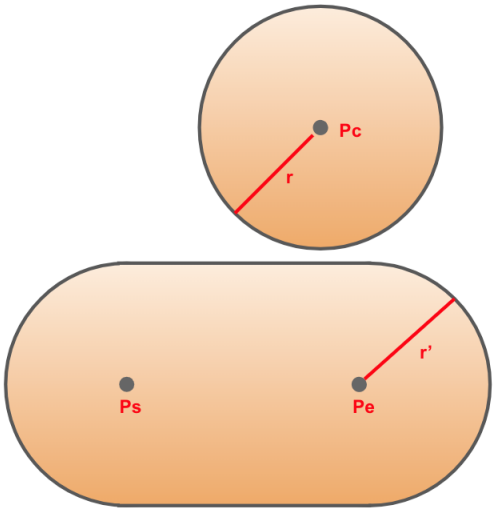
コライダーでよく使われるカプセル(線分ボリューム)コライダーについて

こんにちは、チョコミントです。 今回はカプセルコライダーについてお話ししようと思います。

簡単に説明すると上の写真のような形をしているコライダーです。

線分ボリュームとは

カプセルのことを別名で「線分ボリューム」とも言います。 なぜ「線分ボリューム」と呼ばれるのかというと、線分の全方向に対して指定した半径分線分を 太らせればカプセルが完成するからです。



上の写真を見ていただくとPsとPeを結ぶ線分に対して全方向にr'太らせればカプセルが完成するのが分かりやすいです。

なぜカプセルコライダーは頻繁に使われるのか

1. 衝突判定の際に処理が軽い
2. 人の形にフィットしやすい

この2つくらいだと思います。 1つずつ簡単に説明していきます。

衝突判定の際に処理が軽い

みなさんは球体型のコライダーは、衝突判定の際に処理が比較的軽いことは知っていますか?? なぜなら点との距離を求めればいいだけだからです。

さっきの話の中ではカプセルは「線分を太らせただけ」でしたよね。 ってことはカプセルは線分との距離を求めればいいだけなんです！！ 処理が軽そうですよね！！ ただこの記事を見てくれているみなさんの中には「線分と点」、「線分と線分」との距離って どうやって求めればいいのかわからない人もいますよね。 そこに関しては、また違う記事で説明します。

人の形にフィットしやすい

ゲームでは人型のプレイヤーや敵などが多く登場します。 単純に形が人型にフィットしやすいからです。

最後に

今回はカプセルについての記事でした。 質問やわからなかったことなどがあればぜひ気軽にコメントしてください！

箱(AABB)と球の当たり判定

箱(AABB)と球の当たり判定　by チョコミント

こんにちは、チョコミントです。
今回は**箱(AABB)と球**の当たり判定についてです。

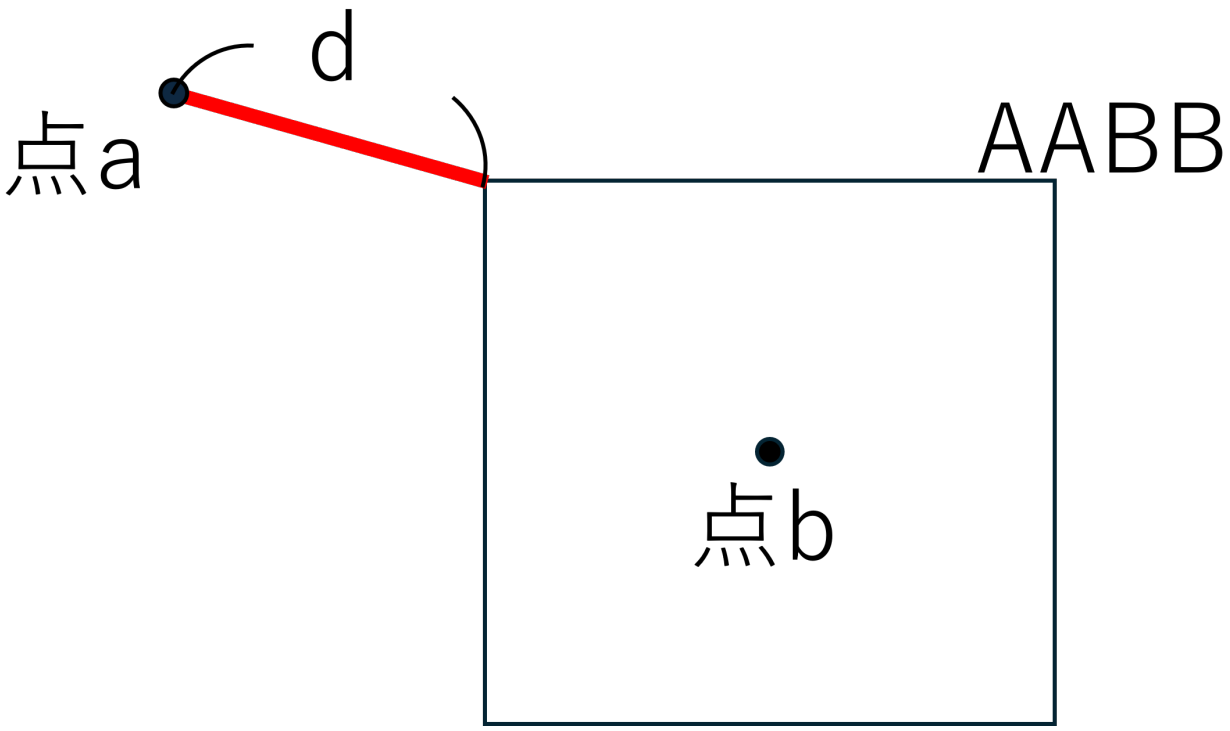
AABB,OBBとは

AABBとOBBについて聞きなれてない人も多いと思うので簡単に説明します。

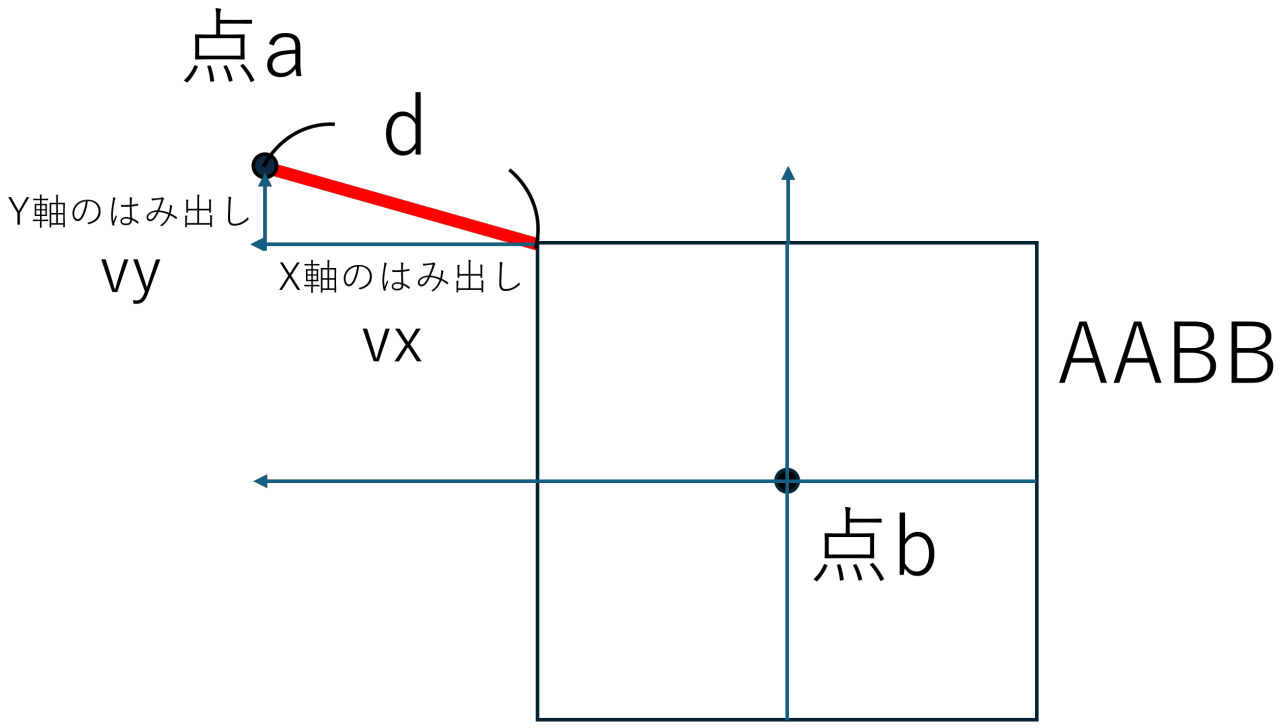
- AABBとは**軸平行境界ボックス**といって**箱の各面の法線が座標軸と平行**なものです。
  箱の横と縦と奥へのベクトルそれぞれがXYZと平行ということですね。
- OBBとは**有向境界ボックス**といって**箱の各面の法線が座標軸と平行ではない**も のです。
  なんとなく理解できたでしょうか。
  AABBとOBBでは**当たり判定をする際の処理速度**がかなり変わってきます。
  OBBについての当たり判定もいつか記事に書く予定です。

AABBと点の距離

さて本題に入っていきましょう。



上の図にある通り、AABBと球の当たり判定を行うには**AABBと点の距離**が分かればいいのです。(図に書いてあるdです)
AABBと点の距離を調べるには、**AABBの各軸(XYZ)で点がどのくらいAABBからはみ出ているか**を調べればいいのです。



上の図ではAABBのX軸、Y軸で点がはみ出ている分、vx、vyでベクトルを作ってみました。
あれこの形どこかで見たことある人いませんか??
そうです、**点と点の距離を求めるのとそっくり**ですね。
つまり**vx、vy、vz(図にはない3Dの場合)の合成ベクトルを求めればAABBと点の距離**になるのです！！簡単ですね！！

```
//AABBの各軸
    XMVECTOR boxX = { 1,0,0 };
    XMVECTOR boxY = { 0,1,0 };
    XMVECTOR boxZ = { 0,0,1 };

    //AABBの各軸の大きさ
    float boxXSize = 0.5f;
    float boxYSize = 0.5f;
    float boxZSize = 0.5f;

    //AABBの中点と点の位置
    XMFLOAT3 boxPos = { 0,0,0 };
    XMFLOAT3 pointPos = { 1,1,1 };

    //各軸での距離を調べる
    float x = fabs(boxPos.x - pointPos.x);
    float y = fabs(boxPos.y - pointPos.y);
    float z = fabs(boxPos.z - pointPos.z);

    //最終的な距離を表すベクトル
    XMVECTOR dis = XMVectorZero();

    //各軸のはみだし距離をdisに加算
    if(x > boxXSize)
        dis += boxX * (x - boxXSize);
    if(y > boxYSize)
        dis += boxY * (y - boxYSize);
    if(z > boxZSize)
        dis += boxZ * (z - boxZSize);

    //disの長さがAABBと点の距離
    return XMVectorGetX(XMVector3Length(dis));
```

サンプルコードです。
ここまでくればAABBと球の当たり判定はちょこっとたすだけです！
AABBと点の距離が球体の半径以下だったら衝突していることになりますね！！

```
//半径以下なら
if (radius >= XMVectorGetX(XMVector3Length(dis))
```

```
{
    //衝突している
}
```

これでAABBと球の当たり判定は完璧です！

最後に

今回はAABBと球の当たり判定についての記事でした。
分からないことや質問などがあれば気軽にコメント待っています！！

2点間の距離

2点間の距離のはなし

ゲームを作っているとよく2次元でも3次元でも2点間の距離を出したくなるよね。
それで登場するのが次の公式

$p_1(x_1,y_1,z_1)$ と $p_2(x_2,y_2,z_2)$ があるときに、その2点の距離は

$$ dist = \sqrt{(x_2-x_1)^2+(y_2-y_1)^2+(z_2-z_1)^2} $$

で計算できる。というもの。
これはこれでよく使うし、計算は合ってます。（結構座標書くときに間違って混乱するよね）

DirectXのプログラム作ってるみなさんでは、ソースコードでは以下の様になると思います。

```
#include <iostream>

using std::cout;
using std::endl;

int main()
{
    //3次元空間上の2点
    XMFLOAT3 v1{ 2.0, 3.0, 4.0 };
    XMFLOAT3 v2{ 4.0, 6.0, 8.0 };

    //距離計算 2点間の距離
    float dist1 = sqrt((v1.x - v2.x) * (v1.x - v2.x)
                + (v1.y - v2.y) * (v1.y - v2.y)
                + (v1.z - v2.z) * (v1.z - v2.z));
    cout << "距離（ふつうの計算）　：" << dist1 << endl;
}
```

XMFLOAT3とかめんどくさい。。。

そんで、DirectXMathに以下のような関数があります。
XMVector3Length 関数

ある2点間の距離は、その2点を始点と終点（どっちがどっちでも同じだよ）とするベクトルの長さに他ならないので、2点をベクトル化してしまえば、この関数が結果を返してくれます。

```
#include <iostream>
#include <DirectXMath.h>

using namespace DirectX;
using std::cout;
using std::endl;

int main()
{
    //3次元空間上の2点
    XMFLOAT3 v1{ 2.0, 3.0, 4.0 };
    XMFLOAT3 v2{ 4.0, 6.0, 8.0 };

    //距離計算 2点間の距離
    float dist1 = sqrt((v1.x - v2.x) * (v1.x - v2.x) + (v1.y - v2.y) * (v1.y - v2.y) + (v1.z - v2.z) * (v1.z - v2.z));
    cout << "距離（ふつうの計算）　：" << dist1 << endl;

    XMVECTOR xv1 = XMLoadFloat3(&v1);
    XMVECTOR xv2 = XMLoadFloat3(&v2);
    float dist2 = XMVectorGetX(XMVector3Length(xv1 - xv2));
```

```
    cout << "距離（ベクトルの長さ）： " << dist2 << endl;
}
```

`XMVector3Length` 関数は戻り値が `XMVECTOR` 型です（距離とか長さなのに）、これは最適化とかメモリ転送の都合でデータの幅を合わせておいて、大きさの揃ったデータだけを使うことで最適化を図っているからです。 戻ってくるデータは `{length, length, length, length}` になっています。 なので、 `x,y,z,w` のどれかをとってやれば長さが取り出せます。

`XMVECTOR` からデータを取り出す命令は

- XMVectorGetX
- XMVectorGetY
- XMVectorGetZ
- XMVectorGetW

などが用意されているのでお好きなものをどうぞ。 これで、距離計算はDirectXMathさえ使っていれば簡単だね。