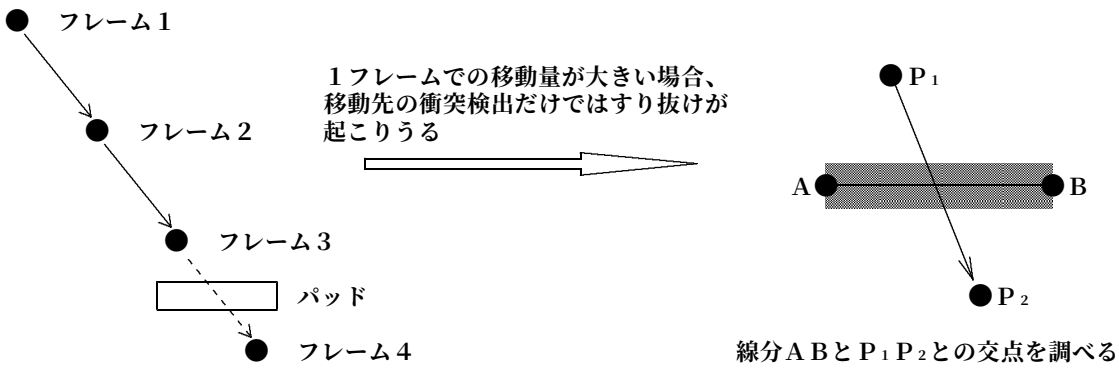
Game Algorithm

▶ 第３回　衝突検出２ ◀

移動を考慮した衝突検出

前回の衝突検出には、少し問題があります。それは、キャラクタが移動していた場合です。スカッシュゲームでは、ボールとパッドが移動していましたが、ボールが高速で移動してきた場合、ボールがパッドをすり抜けてしまう現象が起きてしまいます。これはボールがフレーム単位で離散的に移動するために起こる現象です。これを防ぐには、位置から衝突を検出するのではなく、その移動を示す線分と、パッドの移動を示す線分との交差を調べなければなりません。



▲すり抜け現象とその対策

線分ＡＢと線分Ｐ1Ｐ2の交差を調べる場合、基本的には次のような２段階のステップを踏むことになります。まずふたつの線分のうち片方(ここでは線分ＡＢ)を直線だと仮定します。そしてもう一方の線分(ここでは線分Ｐ1Ｐ2)がこの直線と交差するかどうかを判定します。条件にもよりますが、線分同士の交差判定よりも、直線と線分の交差のほうが計算が楽に済むため、このような方法を使います。そそてここまでが第１段階です。ここでもし交差しないようであれば、このふたつの線分は交差しないので判定処理を終了します。

交差していた場合には第２段階に進みます。次に先ほどとは線分と直線の関係を逆にして同じように交差判定を行います。ここでは線分ＡＢと直線Ｐ1Ｐ2との交差を判定することになります。もし、このケースでも交際が認められる場合、本来の線分ＡＢとＰ1Ｐ2も互いに交わっているということになります。

直線と線分の交差は簡単に判定できると前述しましたが、その簡単な方法は高校で習う数学の考え方に基づいています。すなわち、直線の判別式による判定を使うことで計算を簡単に、そして高速に行うのです。まず直線の式を、

$$Ax + By + C = 0 \quad : \text{式1}$$

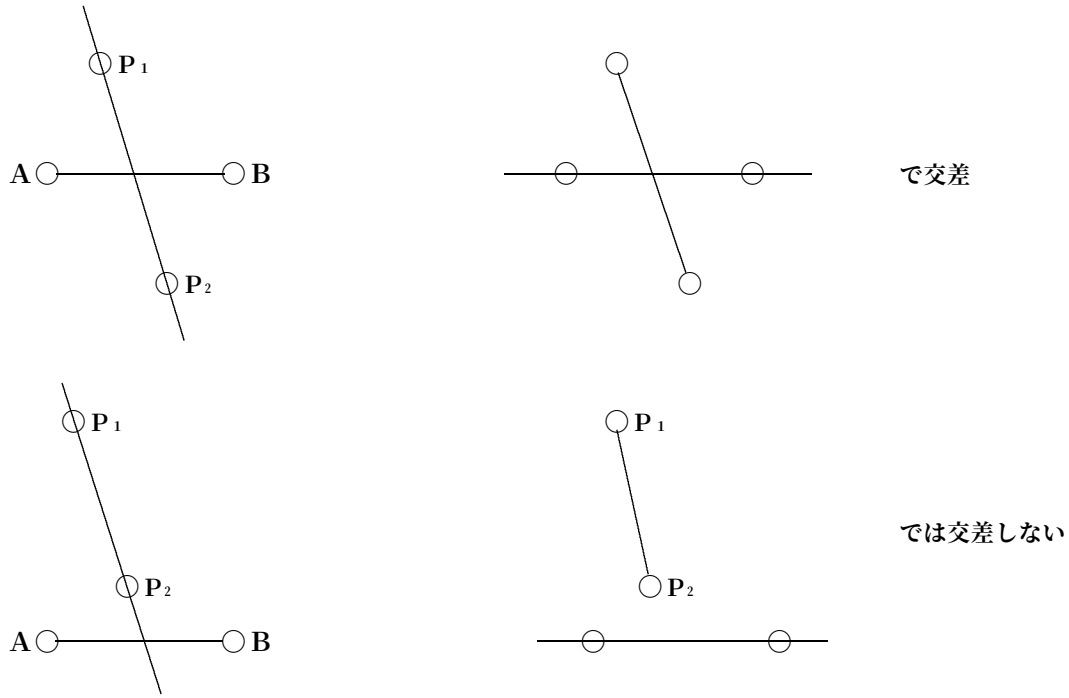
と定義します(Ａ，Ｂ，Ｃは定数)。そしてこの左辺をそのまま使った

$$D(x, y) = Ax + By + C$$

という関数を定義します。式１で示される直線がある場合、平面上の点Ｐ(ｘ，ｙ)はその直線に対して正の領域にあるのか、負の領域にあるのかをこの関数で判別することができます。すなわち、

$D(x, y) > 0$ ： 正の領域
$D(x, y) = 0$ ： 直線上の点
$D(x, y) < 0$ ： 負の領域

となります。線分のふたつの端点が正負側に分かれれば、この線分は直線をまたぐことになるので交差していると判別します。逆に両方とも正、あるいは負になった場合、それは直線に対して同じ側の領域に存在することになり、交差していないことを表します。



▲線分の交差判定

ゲームの中では交差するかどうかがわかるだけでは必要な作業の半分しかこなしたことになりません。そこから今度は交点を求めて、そこでの衝突運動処理(跳ね返りや破壊など)を始めることになるのです。もしゲームシーンのなかで衝突が検出される割合が多いのであれば、上記の判定の第２段階で直接線分と直線の交点を求めてしまい、それが線分上にあるかどうかを調べる方法を使うことで、交点を求める手間を省くという方法もあります。

▼線分の交差判定

```
#include <algorithm>

/*****************************************************************************************************/
/*                                        線分が交差するか調べる                                       */
/*****************************************************************************************************/
bool IntersectLine(POINTD* ptIntersect, const POINTD L1, const POINTD L2,
                   const POINTD M1, const POINTD M2)
{
    double   D;

    // 直線Lの方程式の係数を得る
    double   A1, B1, C1;
    GetLinearEquation(A1, B1, C1, L1, L2);

    // 直線Lと線分Mが交差するか調べる
    D = GetLinearDiscriminant(A1, B1, C1, M1) * GetLinearDiscriminant(A1, B1, C1, M2);
    if(D > 0.0)
        return false;   // 交差しない

    // 直線Mの方程式の係数を得る
    double   A2, B2, C2;
    GetLinearEquation(A2, B2, C2, M1, M2);

    // 直線Mと線分Lが交差するか調べる
    D = GetLinearDiscriminant(A2, B2, C2, L1) * GetLinearDiscriminant(A2, B2, C2, L2);
    if(D > 0.0)
        return false;   // 交差しない

    if(0 != A1 * B2 - A2 * B1) {
        // ２線分が平行でない
        if(NULL != ptIntersect) {
            // 交点を求める
            ptIntersect->x = (B1 * C2 - B2 * C1) / (A1 * B2 - A2 * B1);
```

```
            ptIntersect->y = (A1 * C2 - A2 * C1) / (A2 * B1 - A1 * B2);
        }
    } else {
        // ２線分が平行
        if(max(L1.x, L2.x) < min(M1.x, M2.x) ||
           max(L1.y, L2.y) < min(M1.y, M2.y) ||
           max(M1.x, M2.x) < min(L1.x, L2.x) ||
           max(M1.y, M2.y) < min(L1.y, L2.y))
            return false;   // 交差しない

        if(NULL != ptIntersect) {
            // 交点を求める
            if(IsRange(M1, M2, L1))
                *ptIntersect = L1;
            else if(IsRange(L1, M2, M1))
                *ptIntersect = M1;
            else if(IsRange(L1, M1, M2))
                *ptIntersect = M2;
        }
    }

    return true;
}

/*****************************************************************************************************/
/*                                        直線の方程式を求める                                       */
/*****************************************************************************************************/
void GetLinearEquation(double& A, double& B, double& C, const POINTD pt1, const POINTD pt2)
{
    A = pt1.y - pt2.y;
    B = pt2.x - pt1.x;
    C = pt1.x * pt2.y - pt2.x * pt1.y;
}

/*****************************************************************************************************/
/*                                        直線の判別式を求める                                       */
/*****************************************************************************************************/
double GetLinearDiscriminant(const double A, const double B, const double C, const POINTD pt)
{
    return A * pt.x + B * pt.y + C;
}

/*****************************************************************************************************/
/*                                        点が線分上にあるか調べる                                   */
/*****************************************************************************************************/
bool IsRange(const POINTD L1, const POINTD L2, const POINTD pt)
{
    if(min(L1.x, L2.x) > pt.x || pt.x > max(L1.x, L2.x) ||
       min(L1.y, L2.x) > pt.y || pt.y > max(L1.y, L2.y))
        return false;   // 範囲外

    return true;        // 範囲内
}
```

§ 課題 §

次のプログラムは、スカッシュゲームを改良して作成した「ブロック崩し」ですが、ブロックの数、ボールの跳ね返り、衝突検出といった部分が簡潔に作られています。それらの部分を改良し、完成度を高めましょう。

```
/*
================================================================================
                                ブロック崩し
    Programmed by Hibikino software. Copyright (c) 2003 Hibikino software. All rights reserved.
================================================================================

  【対象ＯＳ】
     Microsoft Windows98/2000以降

  【コンパイラ】
     Microsoft VisualC++ 6.0J ServicePack5

  【プログラム】
     Block.cpp
                ブロック崩し

  【履歴】
     * Version    1.00       2003/08/16  00:00:00  初版

================================================================================
*/

/*****************************************************************************/
/*                              インクルードファイル                         */
/*****************************************************************************/
#include "GameFunc.h"
#include "DDUtils.h"
#include "DIUtils.h"
#include "DXAUtils.h"

#include <math.h>

/*****************************************************************************/
/*                                構造体定義                                 */
/*****************************************************************************/
// 座標(double)
struct POINTD {
    double   x;
    double   y;
};

// 領域(double)
struct RECTD {
    double   left;
    double   top;
    double   right;
    double   bottom;
};

// パッド
struct PAD {
    int      width;     // 幅
    RECTD    rect;      // 領域
};

// ボール
struct BALL {
    POINTD   pos;       // 座標
    double   r;         // 半径
    double   speed;     // スピード
    double   dir;       // 移動角度
    double   add_x;     // ｘ方向増分
    double   add_y;     // ｙ方向増分
};

// ブロック
struct BLOCK {
    RECTD   rect;       // 領域
    int     last;       // 耐久力
};

/*****************************************************************************/
/*                                  定　数                                   */
/*****************************************************************************/
```

```
static const DWORD    FPS         = 60;             // フレーム数
static const double   FRAME_DIRAY = 1000.0 / FPS;   // １フレーム周期

static const double   PI          = 3.14159265358979323846264338327950;

static const int      WALL        = 16;
static const RECT     MOVENABLE   = {0 + WALL, 32 + WALL, 640 - WALL, 480};

/*****************************************************************************************************/
/*                                          外部参照変数                                             */
/*****************************************************************************************************/
extern GameFuncPtr   GameFunc;       // ゲーム関数へのポインタ

/*****************************************************************************************************/
/*                                          グローバル変数                                           */
/*****************************************************************************************************/
static int       g_nScore      = 0;     // スコア
static int       g_nHighScore  = 0;     // ハイスコア

static PAD       g_Pad;                 // パッド
static BALL      g_Ball;                // 球
static BLOCK     g_Block[1];            // ブロック

static int       g_nCnt        = 0;     // 汎用カウンタ

/*****************************************************************************************************/
/*                                      プロトタイプ(プライベート)                                   */
/*****************************************************************************************************/
static void BlockProc();

static void MovePad(PAD& pad, const BYTE byKeyState[]);
static void MoveBall(BALL& ball);
static void SetBallDir(BALL& ball, const double dDir);
static void CollisionDetection(PAD& pad, BALL& ball, BLOCK block[]);

static void DrawSquash(const HDC hDC);
static void DrawBG    (const HDC hDC);
static void DrawWall  (const HDC hDC);
static void DrawPad   (const HDC hDC, const PAD& pad);
static void DrawBall  (const HDC hDC, const BALL& ball);
static void DrawBlock (const HDC hDC, BLOCK block[]);

static void GameOver();
static void GameOverProc();

/*****************************************************************************************************/
/*                                          ブロック崩し                                             */
/*****************************************************************************************************/
void Block()
{
    g_nScore = 0;

    // パッド初期化
    g_Pad.width       = 60;
    g_Pad.rect.left   = 290.0;
    g_Pad.rect.top    = 460.0;
    g_Pad.rect.right  = g_Pad.rect.left + g_Pad.width;
    g_Pad.rect.bottom = g_Pad.rect.top  + 2.0;

    // ボール初期化
    g_Ball.pos.x = 320.0;
    g_Ball.pos.y = 430.0;
    g_Ball.r     =  10.0;
    g_Ball.speed =   1.0;
    SetBallDir(g_Ball, -60.0);

    // ブロック初期化
    g_Block[0].rect.left   = MOVENABLE.left;
    g_Block[0].rect.top    = 360;
    g_Block[0].rect.right  = g_Block[0].rect.left + 32;
    g_Block[0].rect.bottom = g_Block[0].rect.top  + 16;
    g_Block[0].last        = 1;

    GameFunc = BlockProc;   // 初期化が終わったら、メインループ関数へ
```

```
}

/*****************************************************************************************************/
/*                                          ブロック崩しメイン処理                                   */
/*****************************************************************************************************/
void BlockProc()
{
    // キー入力
    BYTE   byKeyState[256];
    DIGetKeyboardState(byKeyState);

    // 座標計算処理
    MovePad(g_Pad, byKeyState);
    MoveBall(g_Ball);

    // 衝突判定
    CollisionDetection(g_Pad, g_Ball, g_Block);

    // 画面構築
    DDColorFill(DDS_BACKBUF, NULL, RGB(0, 0, 0));
    const HDC   hDC = DDGetDC(DDS_BACKBUF);
    DrawSquash(hDC);
    DDReleaseDC(DDS_BACKBUF);

    DDFlip();
    WaitFrame();

    // ゲームオーバー判定
    if(g_Ball.pos.y >= 480.0)
        GameFunc = GameOver;
}

/*****************************************************************************************************/
/*                                             パッド移動                                            */
/*****************************************************************************************************/
void MovePad(PAD& pad, const BYTE byKeyState[])
{
    // 増分設定
    double   dAdd = 0.0;
    if(0 != (byKeyState[DIK_LEFT  ] & 0x80))
        dAdd -= 2.0;
    if(0 != (byKeyState[DIK_RIGHT ] & 0x80))
        dAdd += 2.0;
    if(0 != (byKeyState[DIK_LSHIFT] & 0x80))
        dAdd *= 2.0;

    pad.rect.left  += dAdd;
    pad.rect.right += dAdd;
}

/*****************************************************************************************************/
/*                                              球移動                                               */
/*****************************************************************************************************/
void MoveBall(BALL& ball)
{
    ball.pos.x += ball.add_x;
    ball.pos.y += ball.add_y;
}

/*****************************************************************************************************/
/*                                             衝突検出                                              */
/*****************************************************************************************************/
void CollisionDetection(PAD& pad, BALL& ball, BLOCK block[])
{
    const double   UP_SPEED = 0.02;     // スピード増加量
    RECT           rcIntersect;

    // 壁とパッド
    if(MOVENABLE.left > pad.rect.left) {
        pad.rect.left  = MOVENABLE.left;
        pad.rect.right = pad.rect.left  + pad.width;
    } else if(MOVENABLE.right < pad.rect.right) {
        pad.rect.right = MOVENABLE.right;
        pad.rect.left  = pad.rect.right - pad.width;
```

```
    }

    // 壁と球
    const double   dDir = ball.dir;
    if(MOVENABLE.left >= ball.pos.x - ball.r || MOVENABLE.right <= ball.pos.x + ball.r) {
        SetBallDir(ball, 180.0 - dDir);
        ball.speed += UP_SPEED;
    } else if(MOVENABLE.top >= ball.pos.y - ball.r) {
        SetBallDir(ball, -dDir);
        ball.speed += UP_SPEED;
    }

    // 球とパッド
    RECT   rcBall;
    rcBall.left   = (int)(ball.pos.x - ball.r);
    rcBall.top    = (int)(ball.pos.y - ball.r);
    rcBall.right  = (int)(ball.pos.x + ball.r);
    rcBall.bottom = (int)(ball.pos.y + ball.r);

    RECT   rcPad;
    rcPad.left   = (int)pad.rect.left;
    rcPad.top    = (int)pad.rect.top;
    rcPad.right  = (int)pad.rect.right;
    rcPad.bottom = (int)pad.rect.bottom;

    if(IntersectRect(&rcIntersect, &rcPad, &rcBall)) {
        SetBallDir(ball, -ball.dir);
        g_nScore   += 1;
        ball.speed += UP_SPEED;
    }

    // 球とブロック
    if(0 < block[0].last) {
        RECT    rcBlock;
        rcBlock.left   = (int)block[0].rect.left;
        rcBlock.top    = (int)block[0].rect.top;
        rcBlock.right  = (int)block[0].rect.right;
        rcBlock.bottom = (int)block[0].rect.bottom;

        if(IntersectRect(&rcIntersect, &rcBlock, &rcBall)) {
            ball.speed += UP_SPEED;
            block[0].last--;
            if(0 >= block[0].last)
                g_nScore += 10;
        }
    }
}

/*****************************************************************************************************/
/*                                          球移動方向設定                                           */
/*****************************************************************************************************/
void SetBallDir(BALL& ball, const double dDir)
{
    // 角度を-360.0 ～ 360.0の範囲にする
    const int   w = (int)dDir / 360;
    ball.dir      = dDir - w * 360.0;

    ball.add_x = ball.speed * cos(ball.dir * PI / 180.0);
    ball.add_y = ball.speed * sin(ball.dir * PI / 180.0);
}

/*****************************************************************************************************/
/*                                          スカッシュ描画                                           */
/*****************************************************************************************************/
void DrawSquash(const HDC hDC)
{
    // ライン(ペン)設定
    HPEN   hPen    = CreatePen(PS_SOLID, 1, RGB(255, 255, 255));
    HPEN   hOldPen = (HPEN)SelectObject(hDC, hPen);

    DrawBG   (hDC);             // 文字描画
    DrawWall (hDC);             // 壁描画
    DrawBall (hDC, g_Ball);     // 球描画
    DrawPad  (hDC, g_Pad);      // パッド描画
```

```
    DrawBlock(hDC, g_Block);    // ブロック描画

    // ライン(ペン)解放
    SelectObject(hDC, hOldPen);
    DeleteObject(hPen);
}

/*****************************************************************************************************/
/*                                          背景描画                                                 */
/*****************************************************************************************************/
void DrawBG(const HDC hDC)
{
    // 文字列描画
    SetTextColor(hDC, RGB(255, 255, 255));
    SetBkMode(hDC, TRANSPARENT);

    LPCTSTR   TITLE = "Block Demolition Game";
    TextOut(hDC, 0, 0, TITLE, lstrlen(TITLE));

    TCHAR   szText[256];
    wsprintf(szText, "High Score : %4d      Score : %4d", g_nHighScore, g_nScore);
    TextOut(hDC, 200, 0, szText, lstrlen(szText));
}

/*****************************************************************************************************/
/*                                          壁描画                                                   */
/*****************************************************************************************************/
void DrawWall(const HDC hDC)
{
    Rectangle(hDC, MOVENABLE.left - WALL, MOVENABLE.top - WALL, MOVENABLE.right + WALL, MOVENABLE.top);
    Rectangle(hDC, MOVENABLE.left - WALL, MOVENABLE.top,        MOVENABLE.left,         480);
    Rectangle(hDC, MOVENABLE.right,       MOVENABLE.top,        MOVENABLE.right + WALL, 480);
}

/*****************************************************************************************************/
/*                                          パッド描画                                               */
/*****************************************************************************************************/
void DrawPad(const HDC hDC, const PAD& pad)
{
    Rectangle(hDC, (int)pad.rect.left, (int)pad.rect.top, (int)pad.rect.right, (int)pad.rect.bottom);
}

/*****************************************************************************************************/
/*                                          球描画                                                   */
/*****************************************************************************************************/
void DrawBall(const HDC hDC, const BALL& ball)
{
    MoveToEx(hDC, (int)(ball.pos.x + ball.r), (int)ball.pos.y, NULL);
    AngleArc(hDC, (int)ball.pos.x, (int)ball.pos.y, (int)ball.r, 0, 360);
}

/*****************************************************************************************************/
/*                                          ブロック描画                                             */
/*****************************************************************************************************/
void DrawBlock(const HDC hDC, BLOCK block[])
{
    // 塗りつぶし(ブラシ)設定
    LOGBRUSH   lbr;
    lbr.lbStyle = BS_SOLID;
    lbr.lbColor = RGB(0, 0, 255);
    lbr.lbHatch = 0;

    HBRUSH   hBrush    = CreateBrushIndirect(&lbr);
    HBRUSH   hOldBrush = (HBRUSH)SelectObject(hDC, hBrush);

    if(0 < block[0].last)
        Rectangle(hDC, (int)block[0].rect.left, (int)block[0].rect.top,
                  (int)block[0].rect.right, (int)block[0].rect.bottom);

    // 塗りつぶし(ブラシ)解放
    SelectObject(hDC, hOldBrush);
    DeleteObject(hBrush);
}
```

```
/*******************************************************************************************************/
/*                                                ゲームオーバー                                       */
/*******************************************************************************************************/
void GameOver()
{
    g_nCnt = 0;

    // ハイスコア処理
    if(g_nHighScore < g_nScore)
        g_nHighScore = g_nScore;

    GameFunc = GameOverProc;
}

/*******************************************************************************************************/
/*                                            ゲームオーバー処理                                       */
/*******************************************************************************************************/
void GameOverProc()
{
    // 画面構築
    DDColorFill(DDS_BACKBUF, NULL, RGB(0, 0, 0));
    const HDC   hDC = DDGetDC(DDS_BACKBUF);
    DrawSquash(hDC);
    TextOut(hDC, 278, 232, "Game Over", 9);
    DDReleaseDC(DDS_BACKBUF);

    DDFlip();
    WaitFrame();

    // ５秒たったらメイン処理へ
    g_nCnt++;
    if(FPS * 5 <= g_nCnt)
        GameFunc = Block;
}
```