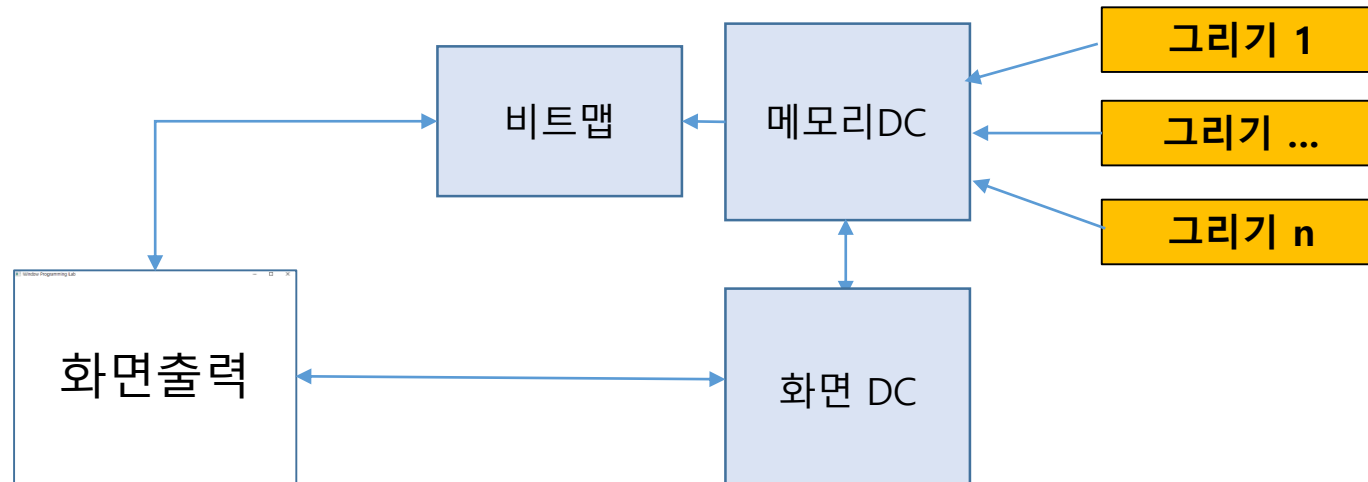


화면이 깜빡일 때 더블 버퍼링 사용하기

2025년 1학기 윈도우 프로그래밍

더블 버퍼링이란

- 애니메이션 출력할 때,
 - 이미지의 잦은 출력으로 화면이 깜박거리는 문제점 발생
 - 메모리 디바이스 컨텍스트를 사용하여 그리기를 원하는 그림들을 모두 출력한 다음, 화면 디바이스 컨텍스트로 한꺼번에 옮기는 방법
 - 추가된 메모리 디바이스 컨텍스트 → 추가된 버퍼 → 더블 버퍼링



메모리 Device Context란

- 메모리 DC:
 - DC의 특성을 가지고 있지만 출력장치와는 연결이 안된 DC
- GetDC를 사용해서 얻은 화면 DC는
 - 출력 장치에 연결된 DC → 출력 대상과 일치하는 크기의 비트맵 객체가 연결
 - 함수 CreateCompatibleDC를 사용해서 얻은 메모리 DC는
 - 출력 대상이 없는 상태로 그리기 특성만 정해져서 만들어지기 때문에 비트맵 객체가 연결되어 있지만 1비트 색상에 폭과 높이가 1인 비트맵이다
 - 따라서, 함수 CreateCompatibleDC로 생성한 메모리 DC를 사용하려면 비트맵 객체를 만들어서 연결하는 작업을 먼저 해야한다.

사용 함수들

HDC **CreateCompatibleDC** (HDC hdc);

- 주어진 DC와 호환되는 메모리 DC를 생성해 준다.
- 주어진 DC가 사용하는 출력장치의 종류나 출력장치가 사용중인 그래픽 드라이버 정보를 가지고 새로운 DC 를 만든다.
- hdc와 동일한 방법으로 그림을 그리지만 화면에 출력은 되지 않는다.
 - HDC hdc: 주어진 DC

HBITMAP **CreateCompatibleBitmap** (HDC hdc, int nWidth, int nHeight);

- hdc와 호환되는 비트맵을 생성하여 반환하는 함수
- 화면 DC와 호환되게 만들어야 한다. (메모리 DC와 호환되게 만들면 1비트 색상수를 사용하는 비트맵을 생성하게 된다.)
- CreateCompatibleDC로 생성한 DC를 사용하려면 비트맵 객체를 만들어서 연결하여 사용
- 생성된 비트맵은 hdc와 호환되는 어떤 메모리 DC에서도 선택되어질 수 있다.
 - HDC hdc: DC 핸들
 - int nWidth/nHeight: 작성하는 비트맵의 가로/세로 사이즈

BOOL **BitBlt** (HDC hdc, int nXD, int nYD, int nW, int nH, HDC memdc, int nXS, int nYS, DWORD dwRop);

- DC 간의 영역 고속 복사
- 메모리 DC의 표면에 그려져 있는 비트맵을 화면 DC로 복사하여 비트맵을 화면에 출력
 - HDC hdc: 복사 대상 DC
 - Int nXD, int nYD: 복사 대상의 x, y 좌표 값
 - Int nW, int nH: 복사 대상의 폭과 높이
 - HDC memdc: 복사 소스 DC
 - int nXS, int nYS: 복사 소스의 좌표
 - DWORD dwRop: 래스터 연산 방법
 - **SRCCOPY**: 소스값을 그대로 칠한다.

사용 방법

LRESULT CALLBACK WndProc (HWND hWnd, UINT iMsg, WPARAM wParam, LPARAM lParam)

```
{
    HDC hDC, mDC;
    HBITMAP hBitmap;
    RECT rt;

    switch (iMsg) {
    case WM_PAINT:
        GetClientRect (hWnd, &rt);
        hDC = BeginPaint (hWnd, &ps);
        mDC = CreateCompatibleDC (hDC);
        hBitmap = CreateCompatibleBitmap (hDC, rt.right, rt.bottom);
        SelectObject(mDC, (HBITMAP) hBitmap);

        //--- 모든 그리기를 메모리 DC에 한다.
        Rectangle (mDC, 0, 0, rt.right, rt.bottom);
        DrawBlock (mDC, blockYnum + 1, blockXnum);
        Rectangle (mDC, rect.left, rect.top, rect.right, rect.bottom);
        Ellipse (mDC, s_ellipse.left, s_ellipse.top, s_ellipse.right, s_ellipse.bottom);

        //--- 화면에 비어있기 때문에 화면 가득히 사각형을 그려 배경색으로 설정하기
        //--- 필요한 그림 그리기 1
        //--- 필요한 그림 그리기 2
        //--- 필요한 그림 그리기 3

        //--- 마지막으로 메모리 DC의 내용을 화면 DC로 복사한다.
        BitBlt (hDC, 0, 0, rt.right, rt.bottom, mDC, 0, 0, SRCCOPY);

        DeleteDC (mDC);
        DeleteObject (hBitmap);

        EndPaint(hWnd, &ps);
        break;

    case WM_TIMER:
        InvalidateRect(hWnd, NULL, false);
        break;

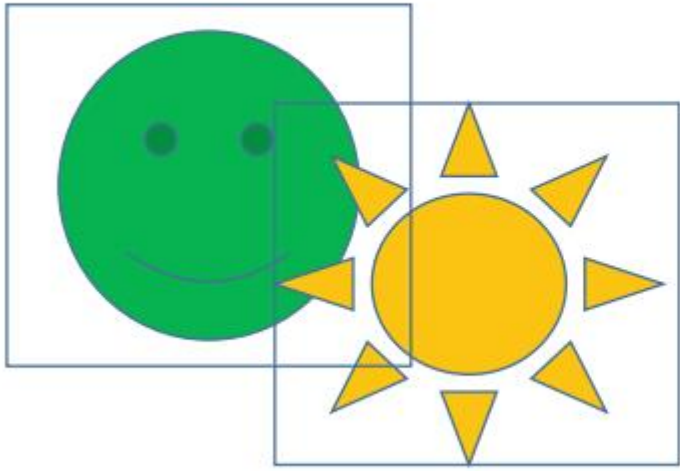
    ...
}
```

2차원에서의 충돌 체크

2025년 1학기 윈도우 프로그래밍

충돌 체크 검사

- 물체간의 충돌이 일어났는지 검사하고 충돌이 일어났을 때 처리하는 프로그램
 - 충돌 영역은 대개 원 또는 사각형으로 설정한다

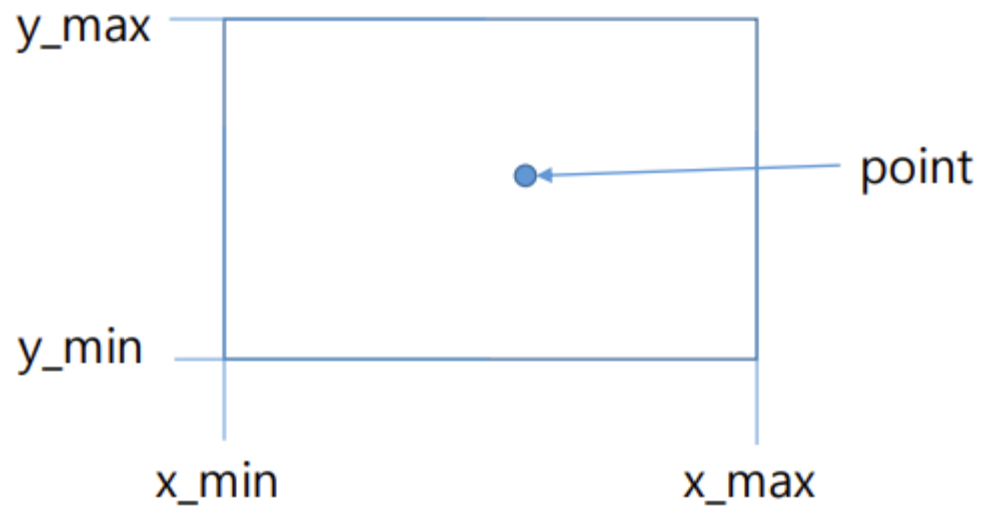


점 대 점 충돌 체크

- 점을 기준으로 비교
 - 두 점이 같은 경우
 - if ((point1.x == point2.x) && (point1.y == point2.y)) → 충돌
 - 두 점의 거리가 일정 영역 안에 있는 경우
 - if (distance (point1, point2) < threshold_value) → 충돌
- 이때, distance : 두 점 간의 거리
threshold_value: 정해진 일정 영역

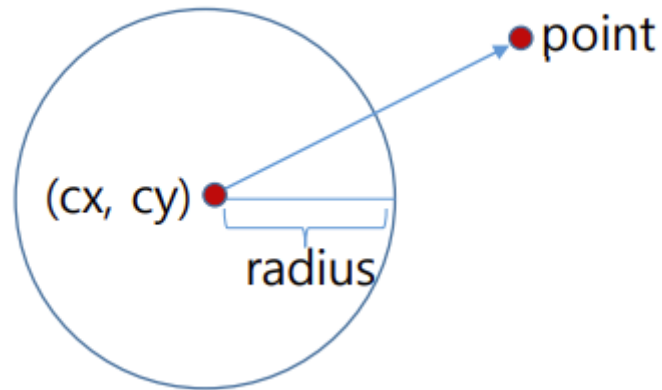
점 대 사각형 충돌 체크

- 점이 사각형의 내부에 있는 경우
 - if ((point.x < x_max) && (x_min < point.x)
&& (point.y < y_max) && (y_min < point.y)) → 충돌



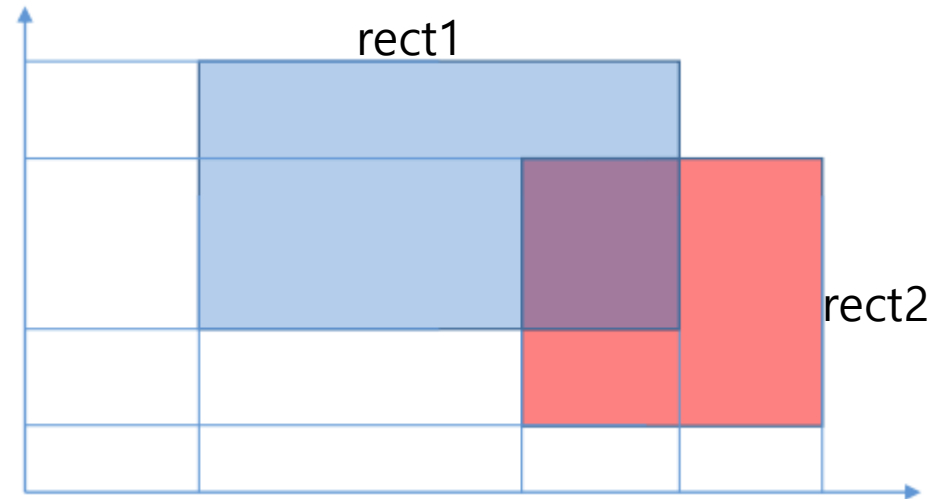
점 대 원 충돌체크

- 점이 원 내부에 있는 경우
 - 원의 중심과 점과의 거리가 원의 반지름 (radius)보다 작은 경우 → 충돌
 - 원의 중심과 점과의 거리: $\sqrt{(\text{point.x} - \text{cx})^2 + (\text{point.y} - \text{cy})^2}$
- $\sqrt{(\text{point.x} - \text{cx})^2 + (\text{point.y} - \text{cy})^2} < \text{radius} \rightarrow \text{충돌}$



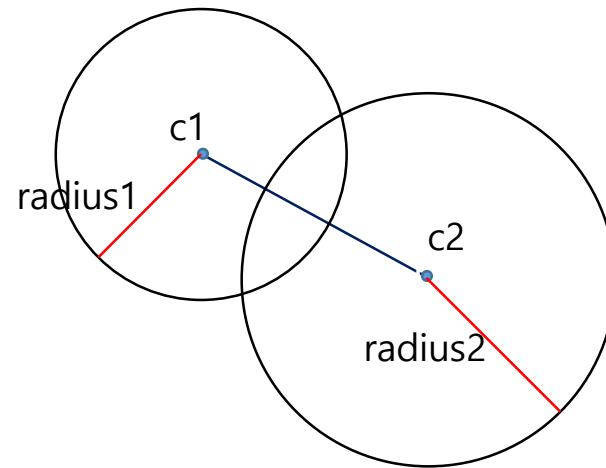
사각형 대 사각형 충돌체크

- 두 사각형의 가장자리를 비교한다.
 - rect1의 left가 rect2의 right보다 작아야 한다.
 - rect1의 top이 rect2의 bottom보다 작아야 한다.
 - rect1의 right가 rect2의 left보다 커야 한다.
 - rect1의 bottom이 rect2의 top보다 커야 한다.
 - if ((rect1.left < rect2.right) && (rect1.top < rect2.bottom) && (rect1.right > rect2.left) && (rect1.bottom > rect2.top)) → 충돌
- 이 때, 첫 번째 사각형: rect1,
두 번째 사각형: rect2



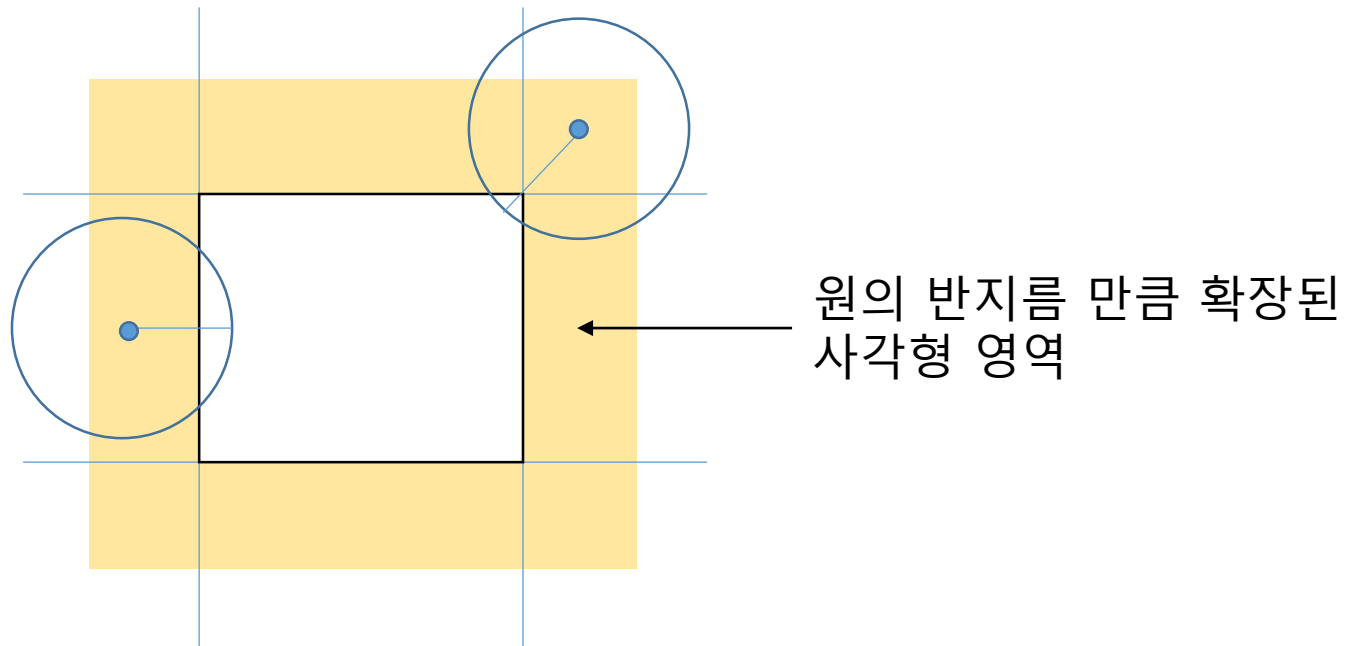
원 대 원 충돌체크

- 두 원의 중심 사이의 거리를 비교한다.
 - 두 원의 중심간의 거리가 반지름의 합과 비교한다.
 - $\sqrt{(c1.x - c2.x)^2 + (c1.y - c2.y)^2} < (radius1 + radius2) \rightarrow$ 충돌



사각형 대 원 충돌체크

- 사각형에 대하여 영역에 따라 검사
 - 사각형을 원의 반지름만큼 확장
 - 원의 중심점이 그 사각형 안에 있다면 → 충돌
 - 원이 사각형 대각선 쪽에 있을 때, 사각형의 꼭지점이 원안에 있다면 → 충돌



대표적 충돌체크 알고리즘

- AABB (Axis-Aligned Bounding Box) 알고리즘
 - 축으로 정렬된 경계 상자 알고리즘
 - 각 축에 대하여 겹쳐지는지 검사하여 충돌체크
 - 간단하게 검사가 가능
 - 물체가 회전하게 되는 경우에는 충돌 경계 상자의 범위가 커지므로 정확한 충돌체크가 어렵다.
- OBB (Oriented Bounding Box) 알고리즘
 - 방향성이 있는 경계 상자 알고리즘
 - 물체의 방향에 따라 경계 상자의 방향을 바꾼다.
 - 충돌을 검사하기 위하여 많은 연산이 필요



충돌 체크 적용

- 게임에서 객체 간의 충돌 체크는 꼭 필요한 요소
 - 게임의 특성, 객체의 모양 / 객체의 개수 등에 따라 적절한 방법의 알고리즘을 선택하여 적용
 - 부분 충돌 체크:
 - 객체의 일부분 적용: 객체의 머리, 손, 발 등 나눠서 충돌 체크를 적용할 수도 있음
 - 공간의 일부분 적용: 검사해야 할 객체들이 많은 경우에는 공간을 분할하여 특정 공간만 충돌 체크를 적용할 수 도 있다.

충돌 체크 적용

- 유용한 함수
 - BOOL `IntersectRect` (LPRECT `lprcDest`, CONST RECT `*lprcSrc1`, CONST RECT `lprcSrc2`);
 - 두 RECT (`lprcSrc1`, `lprcSrc2`)가 교차되었는지 검사한다.
 - `lprcDest`: 교차된 RECT 부분의 좌표값이 저장된다.
 - BOOL `PtInRect` (CONST RECT `*lprc`, POINT `pt`);
 - 특정 좌표 `pt`가 `lprc` 영역 안에 있는지 검사한다.

유용한 코드

- 원과 점의 충돌

//--- (x1, y1)과 (x2, y2)간의 길이

```
float LengthPts (int x1, int y1, int x2, int y2)
```

```
{
```

```
    return (sqrt((x2-x1)*(x2-x1) +(y2-y1)*(y2-y1)));
```

```
}
```

//--- (x1, y1)과 (x2, y2)의 길이가 반지름보다 짧으면 true, 아니면 false

```
bool InCircle (int x1, int y1, int x2, int y2)
```

```
{
```

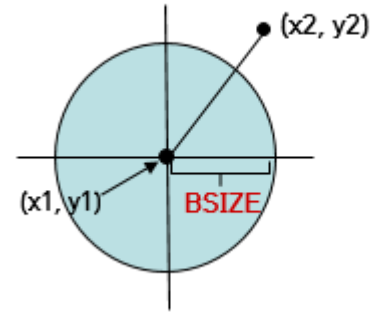
```
    if (LengthPts (x1, y1, x2, y2) < BSIZE)          //--- BSIZE: 반지름
```

```
        return true;
```

```
    else
```

```
        return false;
```

```
}
```



유용한 코드

- 원과 원 충돌

//--- 두 원의 중점 사이의 거리를 구한 후 반지름과 비교

```
struct Circle {  
    int x;  
    int y;  
    int radius;  
}
```

```
bool isCollision (Circle c1, Circle c2)  
{  
    float length = sqrt((((c2.x - c1.x) * (c2.x - c1.x)) + ((c2.y - c1.y) * (c2.y - c1.y))));  
  
    if ( length <= (c2.radius + c1.radius) )  
        return true;  
    else  
        return false;  
}
```