한국컴퓨터정보학회 동계학술대회 논문집 제25권 제1호 (2017. 1)

부드러운 탄막 슈팅 게임 구현을 위한 GDI 와 GDI+ 혼합 사용에 대한 연구

최은호[O], 방정원[*]
[O*]청강문화산업대학교 게임콘텐츠스쿨
e-mail: eunho5751@naver.com[O], jwbang@ck.ac.kr[*]

Study on the use of GDI and GDI + For smooth barrage shooting game Implementation

Eun-Ho Choi[O], Jung-Won Bang[*]
[O*]School of Game, Chungkang College of Cultural Industries

● 요 약 ●

본 논문에서는 부드러운 탄막 슈팅 게임 구현을 위해 GDI와 GDI+ 각각의 장단점을 알아보고, 이 두 가지를 혼용하였을 때 GDI의 단점인 생산성과 안정성, 그리고 GDI+의 단점인 속도 문제까지 해결할 수 있음을 확인하고 이를 이용하여 탄막 슈팅 게임을 구현하는 방법에 대하여 연구하였다.

키워드: GDI(Graphics Device Interface), GDI+(Graphics Device Interface+), barrage shooting game

I. Introduction

Windows API를 이용하여 게임을 개발할 경우 보통 GDI와 GDI+ 중에 한 가지를 선택하여 개발하게 된다. 그러나 빠른 속도로 많은 수의 탄환이 나가는 탄막 슈팅 게임의 경우 화면이 끊기는 등의 구현상의 문제점이 발생하게 된다. GDI와 GDI+ 각각 장단점을 가지고 있기 때문에 GDI와 GDI+를 혼용함으로써 두 가지의 장점만을 취하여 이러한 문제를 해결할 수 없는지 생각해볼 수 있다. 이 논문에서는 GDI, GDI+의 장단점을 각각 알아본 후, 두 가지를 혼용하여 탄막 슈팅 게임을 구현하는 방법에 대하여 알아보고자 한다.

II. Preliminaries

1. Related works

1.1 GDI

GDI의 장점으로는 속도가 있다. GDI를 최초 디자인했던 2~30년 전에는 하드웨어의 성능이 좋지 못하여 화려한 기능보다는 단순하고 빠른 동작이 더 중요했기 때문에 속도를 중점으로 하여 설계된 모듈이다. GDI의 단점으로는 이미지의 확장자를 비트맵이라는 윈도우의 기본 이미지 포맷만 지원한다는 것이다. 비트맵은 압축률이 좋지

않으며 이미지 용량이 너무 커서 부담없이 사용하기 어렵다. 또한 그래픽 속성을 바꿀 때마다 GDI 오브젝트를 일일이 생성 및 선택 후 사용해야 하므로 무척 번거롭고, 사용 후에도 선택 해제 및 파괴를 해줘야 하는 번거로움이 있다. 실수로 생성한 GDI 오브젝트를 파괴하지 않을 경우 리소스가 누출되고 시스템에 치명적인 영향을 끼칠 수 있다. 즉 생산성과 안정성이 떨어지는 단점이 있는 것이다.

1.2 GDI+

GDI+는 GDI와는 대조적으로 GDI의 장점을 단점으로, 단점을 장점으로 가지고 있다. GDI+의 장점으로는 다양한 이미지 포맷을 지원한다. BMP뿐만 아니라 JPG, GIF, PNG 등의 다양한 포맷을 지원하므로 수준 높은 그래픽 처리가 가능하다. 또한 객체지향 수준으로 작성되어 GDI에서 그래픽 처리를 하기위해 했던 GDI 오브젝트를 생성 및 선택하고 해제 및 파괴하는 등의 번거로움이 사라졌다. 즉, GDI와 비교하여 생산성과 안정성이 향상된 것이다. 이런 차이점 외에도 반투명 처리, 안티 앨리어싱 등 많은 기능들을 추가로 제공한다. 단점으로는 GDI와 비교했을 때 그리는 속도가 현저히 떨어졌다. 많은 기능들이 추가되면 단순한 출력에 대해서도 이것 저것 신경써야 할 것이 많아지므로 전체적인 속도가 떨어지기 마련이다. 게임 같은 경우 속도의 저하는 치명적으로 작용하기 때문에 GDI+만으로 게임을 개발하기에는 무리가 있다.

그렇다면 프로그램을 개발할 때 GDI와 GDI+ 중에 어떤 것을 선택해야 할까? 다시 말해 생산성 및 안정성, 그리고 속도 중에 무엇을 택해야 할까? GDI와 GDI+의 장단점은 서로 대조적이니 상호 보완적이라 볼 수 있다. 따라서 답은 둘을 같이 사용하는 것이다.

III. Test Result

1 GDI와 GDI+ 속도 비교

다음은 GDI와 GDI+를 이용해서 비트맵 1000개를 출력하는데 걸린 시간을 출력한 결과이다.

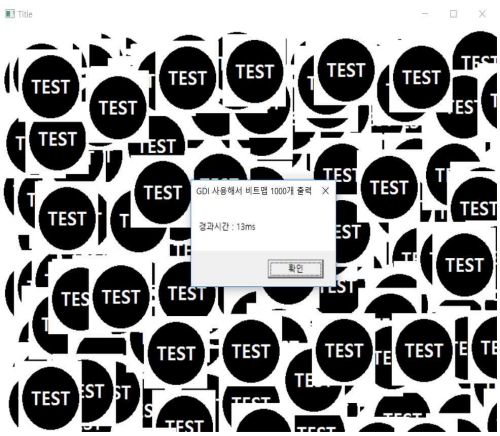

Fig 1. GDI 구현 속도 측정

Fig 2. GDI+ 구현 속도 측정

두 결과를 비교했을 때, GDI의 경우 평균 13ms, GDI+의 경우 평균 440ms가 걸렸다. 즉, 그리는 속도가 3~40배 가량 차이가 났다. GDI+의 경우 눈으로 이미지가 그려지는 모습을 확인할 수 있었다.

2 설계 및 구현

생산성, 안정성을 위해 GDI+를 기본으로 사용하고 속도가 중요시 되는 곳에서만 GDI를 사용하는 것을 기본으로 구현하는 방법을 사용하여 탄막 게임을 구현하였다.

따라서 탄막만 GDI로 그려주고, 다른 모든 부분은 GDI+로 그려주는 방법을 택하였다.

GDI+로 그려줄때는 다음과 같은 명령문 한줄이면 간단히 이미지를 그릴 수 있다.

```
_graphics->DrawImage(m_pImage->image, (int)m_fX, (int)m_fY, (int)width, (int)height);
```

그러나 GDI+를 이용하여 GDI로 변환 후 그려줄 때는 약간 복잡하다.

다음과 같이 Bullet 클래스의 Draw 함수의 매개변수로 Graphics 타입의 객체를 받는다.

```
void CBullet::Draw(Graphics * _graphics)
```

Graphics 객체로부터 DC를 얻어와서, CompatibleDC를 생성한 후, Graphics 객체가 memDC를 가리키도록 한다.

```
HDC hdc = _graphics->GetHDC();

HDC memDC = CreateCompatibleDC(hdc);
```

Graphics* graphics = Graphics::FromHDC(memDC);

HBITMAP을 만든 후, 미리 생성되어 있던 비트맵으로부터 초기화 시킨 후, SelectObject로 선택한다. (memDC로 선택해야 한다.)

HBITMAP hBitmap;

bitmap->GetHBITMAP(Color(0, 0, 0, 0), &hBitmap);

SelectObject(memDC, hBitmap);

hdc에 memDC에 있는 내용을 그려준다. 어떤 Blt 함수를 이용해도 상관 없다.

TransparentBlt(hdc, (int)m_fX, (int)m_fY, (int)(width * m_fScale), (int)(height * m_fScale), memDC, 0, 0, width, height, RGB(255, 255, 255));

메모리 누수가 일어나지 않도록 사용이 끝난 메모리들을 제거.

delete graphics;

DeleteObject(hBitmap);

DeleteDC(memDC);

_graphics->ReleaseHDC(hdc);

Table 1. GDI/GDI+ 구현 코드

```
case WM_PAINT:
{
RECT rect;
PAINTSTRUCT ps;
HBITMAP bitmap;
HDC hdc, memDC;

hdc = BeginPaint(hWnd, &ps);
memDC = CreateCompatibleDC(hdc);
GetClientRect(hWnd, &rect);
bitmap = CreateCompatibleBitmap(hdc,
    rect.right, rect.bottom);
SelectObject(memDC, bitmap);
FillRect(memDC, &rect,
  (HBRUSH)GetStockObject(WHITE_BRUSH));

Graphics graphics(memDC);

Draw_GDIPlus(graphics);
Draw_GDI(graphics);

BitBlt(hdc, 0, 0, rect.right,
rect.bottom, memDC, 0, 0, SRCCOPY);

DeleteObject(bitmap);
DeleteDC(memDC);
EndPaint(hWnd, &ps);

return 0;
}
```

```
void Draw_GDIPlus(Graphics* graphics)
{
Image image(L"C:Wtest.png");

graphics->DrawImage(&image, 0, 0, 100, 50);
}

void Draw_GDI(Graphics* graphics)
{
Bitmap image(L"C:Wtest.bmp");

HDC hdc = graphics->GetHDC();
HDC memDC = CreateCompatibleDC(hdc);

HBITMAP bitmap;
image.GetHBITMAP(Color(0, 0, 0, 0), &bitmap);
SelectObject(memDC, bitmap);

BitBlt(hdc, 0, 0, 100, 50, memDC, 0, 0,
SRCCOPY);

DeleteObject(bitmap);
DeleteDC(memDC);
graphics->ReleaseHDC(hdc);
}
```

다음은 위의 방식으로 구현한 탄막 슈팅 게임의 모습이다.

IV. Conclusions

본 논문에서는 부드러운 탄막 게임 구현을 위해 GDI와 GDI+의 장단점을 확인하고 혼용했을 때 GDI의 속도, GDI+의 생산성과 안정성을 모두 취할 수 있음을 확인하였다. GDI만을 이용할 경우, 속도는 빠르지만 여러가지 번거로움 및 리소스 누출의 위험이 있고, GDI+만을 이용할 경우, 생산성 및 안정성이 향상되지만 그리는 속도가 현저하게 떨어짐을 확인하였다.

References

[1] Microsoft Developer Network (https://msdn.microsoft.com)

[2] Software Engineering
(http://www.soen.kr)