Object Graphics를 소개합니다.

IDL

Direct Graphics vs Object Graphics

오브젝트 그래픽스가 등장하기 전에는 다이렉트 그래픽스라는 말이 없었습니다. 그냥 IDL의 그래픽이라고 하면 다이렉트 그래픽스였습니다. IDL 5.0부터 새로운 그래픽 환경인 오브젝트 그래픽스가 등장하였고 기존의 그래픽 시스템은 다이렉트 그래픽스라고 명명되었습니다.

Object 라는 것

그래픽스 관점에서 Object라는 것은 모든 표출 대상(최소 부품, 부품을 조립한 모델, 모델에 비추는 광선 등)과 이들로 조합된 컨테이너의 총칭이고, 가장 큰 특징은, **이들이 메모리에 존재한다는 것**입니다. '기억해 두었다가 계속 사용하는 것'이 바로 오브젝트입니다.

간단한(?) 사용 절차를 소개합니다.

Direct Graphics와 Object Graphics는 사용법이 다릅니다. 단순함이 미덕인 Direct Graphics와 달리 Object Graphics는 약간의 단계를 밟아야만 그림이 나옵니다.
이 문서의 목적이 Object Graphics를 간단히 "소개"하는 것이므로 좌표변환의 원리에 대해서는 깊이 이해하지 않고 넘어가는 것이 좋겠습니다. 다만, 부품(Atom)→조립품(Model)→표출범위(시야; View) 의 기본 절차를 거쳐, 정해진 표출 범위를 어디로 보내느냐(화면, 파일, 클립보드 등)를 지정해 주어야 하나의 그림을 볼 수 있다는 단계적인 특성과, 이 절차에 관심을 가지고 한줄 한줄 실행해 주십시오.

```
IDL> data=HANNING(400, 400)
IDL> xc=NORM_COORD([0, 399]) & xc[0]=-0.5
IDL> yc=NORM_COORD([0, 399]) & yc[0]=-0.5
IDL> zc=NORM_COORD([0, max(data)]) & zc[0]=-0.5
```

NORM_COORD 사용과 영점조정을 통해 데이터를 -0.5~0.5의 범위로 스케일링 하는 것이 목적입니다. 그렇게만 이해하시고 다음으로 넘어갑니다. 물론 이 부분은 실제 오브젝트 그래픽스를 사용할 때 중요한 문제이고 어려운 내용은 아닙니다만 이 문서의 목적에 벗어납니다. 하지만 이름 원칙은 기억해 두십시오. IDL의 오브젝트 그래픽스 구성요소는 모두 'IDLgr'로 시작합니다. 대소문자는 사실 구별하지는 않습니다. 자, 부품 1번 나옵니다. Surface입니다.

```
IDL> oAtom1=obj_new('IDLgrSurface', data, style=2, $
    XCOORD_CONV=xc, YCOORD_CONV=yc, ZCOORD_CONV=zc)
```

[XYZ]COORD_CONV를 이용해 스케일을 조정합니다. 이 문서의 목적대로 이 부분은 스케일 조정이라고 이해하고 넘어갑니다. 앞에 얘기했듯이 데이터의 범위를 -0.5~0.5로 조정하고자 하는 목적입니다. STYLE 키워드는 그물형, 면처리, 레고형 등으로 고를 수 있게 합니다. 오브젝트 그래픽스는 이것만으로 아무것도 보이지 않습니다. 몇 단계의 절차가 더 필요합니다.

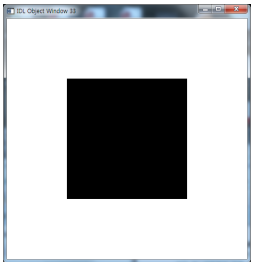
```
IDL> oModel=obj_new('IDLgrModel')        ;모델. (조립품)
IDL> oView=obj_new('IDLgrView')          ;뷰 (표출 범위, 시야)
```

역시 아직 아무것도 나오지 않습니다. 이들을 연결해 주어야 합니다. 있는 부품들을 조립품에 더하고, 완성된 조립품(모델)을 표출 범위(뷰)에 넣어야 합니다.

```
IDL> oModel->add, oAtom1          ;조립품에 부품을 더함
IDL> oView->add, oModel           ;표출 범위에 모델을 넣음
```

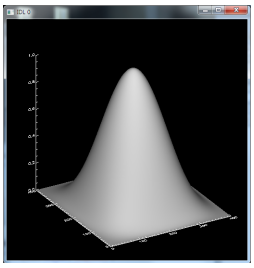
거의 다 왔습니다. 현재 Atom→Model→View 의 단계까지 연결해 놓았습니다. 이제 View를 화면으로 보내겠습니다.

```
IDL> oWindow=obj_new('IDLgrWindow', DIMENSIONS=[500, 500])
IDL> oWindow->draw, oView
```



500*500의 창을 하나 열고 거기에 우리가 그동안 만든 뷰인 oView를 그립니다. 아직, 좀 실망스러울 수 있지만, 어쨌든 그림이 하나 나왔습니다. 여러분은 아마도 다이렉트 그래픽스에 이미 익숙하므로 다음과 같은 명령을 통해 이미 그림을 그려 보았을 거라고 예상합니다.

```
IDL> shade_surf, data
```



shade_surf, data

오브젝트 그래픽스 예제의 결과는 이를 위에서 내려다 본 모양이어서 정사각형으로만 보이고 있습니다. 도대체 왜 오브젝트 그래픽스는 이렇게 단계적 절차로 사용하게 만들어 놓았을까요? 이렇게 복잡한데도 사용을 하려는 이유는 무엇일까요? 오브젝트 그래픽스만의 장점이 있기 때문입니다. "복잡한 것을 그리기 위해서는 오브젝트 그래픽스가 더 직관적이고, 그 결과가 우수하기 때문"입니다. 하나씩 검토해 보겠습니다.

장점 1. 조립의 개념으로 완성(계층적)

오브젝트라는 것은, 지우지 않는 한 메모리에 그대로 남아 있습니다. 두 번째 부품은 조명(Light)입니다..

```
IDL> oAtom2=obj_new('IDLgrLight', LOCATION=[-0.5, -0.5, 1], $
    DIRECTION=[0, 0, 0], TYPE=2)
IDL> oModel->add, oAtom2          ;부품 2번을 기존 모델에 결합
```

조명의 위치와 비추는 방향, 그리고 조명의 종류(2번은 직광입니다. 스포트라이트, 산란광 등이 있습니다)를 지정합니다. 부품 2번도 모델에 포함시켰습니다. 이로서 IDL의 메모리에서는 모든 변화가 반영되었습니다만, 아직 표출은 변화 없이 그대로입니다. 화면을 다시 그려야 합니다.

```
IDL> oWindow->draw, oView
;이 실습의 표출범위는 앞에서 oView 변수로 지정되었으므로...
```

조명발이 조금 반영된 것이 확인되나요? Surface의 색을 순수 빨강으로 바꾸어 보겠습니다.

```
IDL> oAtom1->setProperty, color=[255, 0, 0]
IDL> oWindow->draw, oView
```

장점 2. 속성 변경에 대한 표출 반응 속도

다이렉트 그래픽스가 따라올 수 없는 모델 회전을 한번 감상해 보시죠.

```
IDL> for i=0, 75 do begin & oModel->rotate, [1, 0, 0], -1 &$
```

```
    oWindow->draw, oView & endfor
```

이처럼 모델 변환에 대한 빠른 표출 반응 속도는, GUI에서 사용자의 마우스 조작에 대해 부드러운 반응을 보장하게 됩니다. GUI의 표출로 오브젝트 그래픽스가 주목받는 이유입니다. 실제 GUI 인터페이스에 넣어 회전, 주밍, 이동을 해 볼까요?

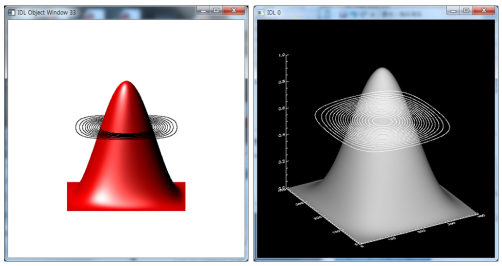
```
IDL> xobjview, oModel
```

장점 3. 태생이 3D 그래픽스

```
IDL> oAtom3=obj_new('IDLgrContour', data, $
       XCOORD_CONV=xc, YCOORD_CONV=yc, $
       planar=1, geomz=0.1, N_LEVELS=20)
IDL> oModel->add, oAtom3
IDL> oWindow->draw, oView
```

부품 3번으로 조촐하게 등치선을 60%쯤의 높이에 만들어 보았습니다(실습에서 데이터의 범위를 -0.5~0.5로 맞추어 놓았으므로 GEOMZ=0.1 이 60% 높이가 됩니다) 이 결과가 별 것 아닌 것처럼 보인다면, 다이렉트 그래픽스와 비교해 봅시다.

```
IDL> Shade_surf, data, /SAVE
IDL> contour, data, NLEVELS=20, zvalue=0.6, /T3D, /OVER
```

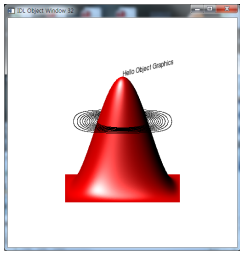


왼쪽 오브젝트 그래픽스와 오른쪽 다이렉트 그래픽스.
Surface의 뒤쪽에 위치하는 선이 제대로
가려지는지(hidden line 처리)에서 차이를 보임

다이렉트 그래픽스는 그리고 나면 아무것도 기억하지 않습니다. 즉, Surface를 그렸는지 기억하지 못한다는 것입니다. 그래서 등치선을 추가로 그릴 때, Surface가 있는지 모른 채 그냥 덮어서 그리게 됩니다. 다이렉트 그래픽스에서도 Z-Buffer를 사용하여 해결할 수 있습니다만, 아무래도 이런 3D 처리는 오브젝트 그래픽 쪽이 훨씬 더 뛰어납니다.

장점 4. 예쁜 폰트 사용

```
IDL> oAtom4=obj_new('IDLgrText', 'Hello Object Graphics', $
     Location=[0, 0, 0.5], UPDIR=[0, 0, 1], baseline=[1, 1, 0])
IDL> oModel->add, oAtom4
IDL> oWindow->draw, oView
```



오브젝트 그래픽스의 장점이라기보다는 다이렉트 그래픽스의 약점입니다. 오브젝트 그래픽스는 비교적 신형 그래픽 체계이므로 트루타입 폰트를 자유롭게 사용할 수 있는 장점이 있습니다. 글자가 써지는 방향(BASELINE)과 글자를 쓰는 평면(UPDIR)을 자유롭게 지정할 수 있는 것도 다이렉트 그래픽스에는 없는 장점입니다. 네 번째 부품인 IDLgrText까지 조립한 채로 모델을 한번 돌려 봅시다. 조립만 제대로 했다면 이런 일은 아주 쉽습니다.

```
IDL> for i=0, 45 do begin & oModel->rotate, [0, 1, 0], -1 &$
       oWindow->draw, oView & endfor
```

장점 5. 표출 방향(장치)의 손쉬운 전환

오브젝트 그래픽스의 표출 대상은 IDL의 메모리에 항상 존재하는 것이고, 이를 어느 쪽으로 내보낼지는 자유롭게 결정할 수 있습니다. 꼭 한쪽으로만 보내야하는 제약도 없습니다. 앞에 열어 두었던 xobjview 창이 아직 있다면 화면을 한번 클릭해 보세요. 지금까지 모델에 대한 변형이 여기에도 똑같이 반영되어 나타날 것입니다. PDF 파일로 보내 볼까요?

```
IDL> oPDF=obj_new('IDLgrPDF')
IDL> oPDF->addPage
IDL> oPDF->draw, oView
IDL> oPDF->save, 'testog.pdf'
```

장점 6. 사실적인 광선처리

앞서 IDLgrLight를 이용하여 다양한 조명을 사용할 수 있음을 보았습니다. 조명 개수에 제한도 없습니다. 얼마든지 필요한 만큼 만들어 모델에 붙이면 됩니다. 오브젝트 그래픽스의 자연스러운 광선처리를 한번 감상하시죠. 다음 명령은 조명의 x 좌표를 -0.5에서 0.02씩 이동시켜 가는 것입니다. 실행하고 화면에 진행되는 변화를 살펴 보세요.

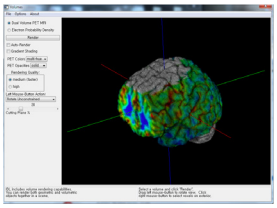
```
IDL> start=-0.5
IDL> for i=0, 200 do begin &$
     oAtom2->setProperty, location=[start, -0.5, 1] &$
     oWindow->draw, oView & start+=0.02 &$
     endfor
```

장점 7. 알파채널을 이용한 투명도 조정

복잡한 구조를 제대로 표현하기 위해서 자유로운 투명도 조절은 필수적입니다.

```
IDL> oAtom3->setProperty, alpha_channel=0.3
IDL> oWindow->draw, oView
```

장점 8. 볼륨 렌더링



3차원 배열 데이터를 볼륨 렌더링하기 위해서는 오브젝트 그래픽스를 사용해야 합니다. 다음 데모를 실행해 보세요.

```
IDL> d_volrendr
```

장점 9. 고성능 그래픽 카드의 성능 반영

오브젝트 그래픽스는 실리콘 그래픽스의 3D 그래픽 엔진 OpenGL을 IDL에서 사용할 수 있도록 만들어 놓은 인터페이스입니다. 그러므로 OpenGL 가속 기능이 있는 그래픽 장치를 사용한다면 오브젝트 그래픽은 더 빠르게 실행됩니다.

보다 쉽게 사용하는 오브젝트 그래픽스

iTools나 New Graphics(IDL 8.0부터)를 사용하면, 다이렉트 그래픽스와 유사한 사용법으로 오브젝트 그래픽스를 사용할 수 있습니다.