Processing

이 튜토리알은 ITP 학생들을 위하여 열었던 워크숍, "Introduction to Computational Media", "Programming for Non-Programmers", 그리고 "Code and Me" 의 보조 교재로 Josh Nimoy 가 쓴 것을 한글로 번역한 것입니다.

Language: English , Español, Japanese , Korean

Workshop Teacher
Josh Nimoy
contact: jn429 [at] nyu [dot] edu
website

Software Creators
Processing is an open project
initiated by
Ben Fry and Casey Reas

Japanese Translation by
Hironobu Fujiyoshi and Ayako Takabatake
contact: hf [at] cs [dot] chubu [dot] ac [dot] jp

Korean Translation by
Koo-Chul Lee
contact: kclee [at] phya [dot] snu [dot] ac [dot] kr

Spanish Translation by
Gerald Kogler & Angela Precht
contact: geraldo [at] yuri [dot] at

소개

프로세싱은 전자 매체가 실현하는 컨셉공간을 만드는 환경을 제공한다. 프로세싱은 전자적 예술작품을 만드는 과정을 통해서 컴퓨터 풀그림을 배울 수 있는 환경이기도 하며 아이디아를 개발할 수 있는 전자 스켓치북이라고도 할 수 있다.
http://www.proce55ing.net/

프로세싱은 우리가 알고 있는한 가장 쉽게 접근할 수 있는 자바 컴파일러이고 쌍방향성 그래픽스와 멀티미디아 풀그림 환경이기도 하다. 이 프로세싱은 호스트 컴퓨터에서 독립적으로 실행시킬 수 있는 작품을 만들 수도 있고 웹 브라우저에 끼워 넣는 자바 애플랫으로도 만들 수 있다. 또 이 시스템은 그래픽스 풀그림 교육(educational graphics programming environments)과 진짜 "자바" 사이의 간격을 메꿔 주기 위한 목적으로도 만들었다. 프로세싱은 풀그림 교육의 도구이기도 하지만 그 이상의 가능성과 발전성은 얼마던지 있다.

이 튜토리알의 목적은 매크로미디아의 Flash 와 Director 의 유저에게 프로세싱과 매크로미디아 제품들과 비교하고 대비하면서 프로세싱을 소개하려는 목적으로 썼다. 이러한 목적의 바탕에는 매크로미디아 제품을 사용하면서 터득한 지식을 잘 활용하면 프로세싱을 배우는데 상당한 시간을 절약할 수 있다고 생각했기 때문이다. 그래서 이 튜토리알은 위에 든 매크로미디아 제품중 하나에 대해서 최소한의 기초지식을 갖춘 독자를 염두에 두고 썼다. 이 튜토리알을 마치면 여러분은 자신의 프로세싱(자바)작품을 만들어 발표하고 BX-24 chip 을 갖춘 시리알 포트를 통해서 통신도 할 수 있을 것이다.

차례

머릿말

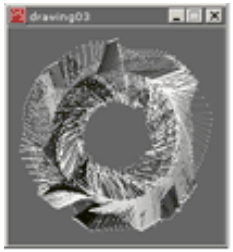
현재 컴퓨터를 쓰는 인터랙티브 디자인 교재에서는 대다수는 플래시 아니면 디렉터를 도구로 쓰고 있다.그런데 학생들은 다른 툴을 쓰는 환경에서 제작된 작품에 영향을 받아 좀더 동적인 기하적 디자인이나 앨고리즘이 복잡한 작품을 만드는 단계에 온다. 프로세싱을 쓰기 전의 일이지만 ITP에서 이런 강좌에서 실험을 하나 해 보았다. Director Lingo를 가르치는 도중에 일주간 자바 폴그림을 가르쳐 보았다. 다른 언어체계를 통해서 Lingo 말고 좀더 다양한 언어세계를 이해하고 시야를 넓혀주기 위해서였다. 템프레이트(template) 하나와 참고문헌(rosetta stone reference)를 주고 폴그림을 바꿔 보도록 하였다. 일주일간의 혼란을 겪은 학생들가운데에는 아무것도 못한채 그저 좀더 자바를 배워야겠다는 생각만 토로하는 경우가 있었다. 그런데 딱한 것은 대학에서 가르치는 일반 자바 강좌는 텍스트모드 콘솔에서 폴그림을 짜고 컴파일하는등 특수한 강좌를 제외하고는 "애플렛 그래픽스"와는 거리가 먼 것들이라는 사실이다. 그러니 학생들에게 그런 자바 강좌를 듣고 오라고 하긴 쉽지 않았다. 그래서 이 강좌는 이런 사정을 감안하여 프로세싱환경에서 대학의 일반 자바강좌와 그래픽스와 디자인 위주의 폴그림 교육간의 갭을 줄여 주려는 것이다. 결코 이 강좌로 자바강좌를 대치하려는 것은 아니다. 자바 언어의 심오한 뉴앙스를 배우기 전 단계로 자바교육의 보충교육쯤으로 알아 두면 좋겠다. 또한 프로세싱과 자바를 매크로미디아 제품의 다음단계로 또는 그 바탕(lower level)시스템으로 내 세우는 것도 아니라는 점에 유의하기 바란다. 프로세싱과 자바는 이들 상업제품과는 다른 내용을 달리 폴그림할 수 있는 단순한 대안일 뿐이다. 여러분이 현재 자바 강의를 듣고 있다면 담당강사가 얼마나 유연한 생각을 갖고 있느냐에 달렸지만 프로세싱을 써서 여러분의 과제를 제출할 수도 있을 것이다. 이 튜토리알은 Casey Reas 와 Ben Fry 가 만든 프로세싱 사이트에서 얻은 정보를 내 나름대로의 관점에서 종합해서 쓴 것이다.

프로세싱 무른모 어얻기

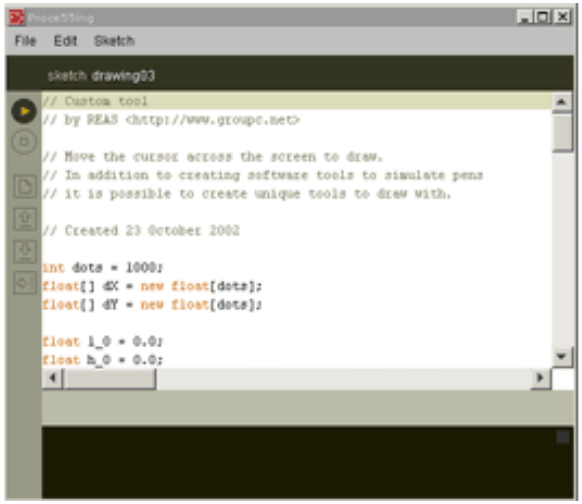
프로세싱은 무료로 아직은 개발 단계에 있다. 개발이 끝난 다음에도 무료로 배포될 것이다. 현재는 알파단계에 있다. 알파버전은 베타 버전보다 앞선 단계다. 버그를 잡고 새 기능들을 추가하고 있는 단계다. 크로스푸랫폼 인스톨 폴그림을 다운로드하기 위해서는 시험사용 코뮤니티에 가입하겠다는 이메일을 개발자에게 보내면 된다. 프로세싱 웹사이트에 가서 Download를 클릭하면 다음 단계에에 대한 지시를 해 준다. 또 테스터 협력자사이에는 메세징 시스템이 있다. 또 프로세싱 사이트에서 Discourse를 클릭하여 이 사이트에 가담하면 다른 테스터들과 정보를 교환하고 어떤 주제이던 도움을 받을 수 있고 줄 수도 있다. 뿐만 아니라 이 사이트에 가담하면 저자들과 직접 대화를 할 수도 있고 이 튜토리알 저자와도 만날 수 있다. 이 사이트에는 끊임없이 업데이트하고 새로운 버전이 올라 오고 있는 역동적인 사이트라는 사실이다.

인터페이스 둘러 보기

아래 그림은 www.Proce55ing.net에 있는 그림이다. 관련사항은 Reference를 클릭하고 Environment.를 클릭하면 볼 수 있다.



Display Window



Menu

Text Editor

Message Area

Text Area

이 그림을 보는 순간 여러분은 즉시 "아니 이렇게 단순한 인터페이스라니? 어떻게 플래시나 디렉터같은 일을 해 낼 수 있단 말인가?" 하는 질문을 할지 모른다. 디렉터와 플래시에는 상업용 멀티미디아에 공통적용되는 온갖 입력장치나 미디아 편집기가 탑재되어 있다. 한편 프로세싱에서는 이런 것을 쓰지 못하고 모두 다른 풀그림으로 하거나 자바로 짜서 쓰고 있다. 예컨데 플래시에는 미니 일러스트레이터(Illustrator) 가 들어 있고 디렉터에는 미니 포토샵 (Photoshop) 이 탑재되어 있다. 그런 연유로 이 두 무른모에서 만든 작품의 많은 부분이 통합편집기능의 제약을 서로 닮고 있을 수밖에 없다. 프로세싱(자바도 마찬가지지만)에서는 여러분이 벡타패쓰(vector paths)나 GIF 파일의 목록을 스스로 만들고 풀그림을 통해서 이들 그림을 구현해야 한다. 따라서 여러분 자신이 형식과 구조를 자유로이 결정하고 스크린위의 픽셀 하나하나를 풀그림 언어를 통해서 직접적으로 제어 통제할 수 있다. 그런 의미에서 현상에 만족하지 않고 가존의 자동 툴을 넘어선 새로운 형식을 탐구하고저 하는 실험정신이 강한 독자에게는 프로세싱이야 말로 아주 편리한 도구라 할 수 있다.

아래에 윈도우 왼쪽에 있는 6개의 버튼을 간략히 설명한다.

play 버튼은 Director 와 Flash의 것과 마찬가지이다. 이 버튼을 누르면 여러분이 입력한 코드가 풀그림으로 되어 실행된다.

stop 버튼역시 Director 와 Flash의 것과 마찬가지이다. 이 버튼을 누르면 실행되던 풀그림이 멈춘다.

이 버튼은 new 버튼인데 새 파일을 만든다. 프로세싱에서는 이 파일들을 *sketches* 라 부른다. 그러나 여러분은 이 파일을 애플렛, 풀그림, 양방향성 작품(interactive pieces)이라 불러도 된다. Director 와 Flash 의 *movies* 에 해당된다.

Opens 버튼이다. 이미 존재하는 sketch 파일을 연다. 이 버튼을 누르면 메뉴창이 팝업된다. 이 팝업창에는 sketch folder 에 저장되어 있는 여러분의 기존 작업물들을 골라 열 수 있다. 또 example sketches 디렉토리에 가면 유명한 새 미디아 디자이너/아티스트의 작품들을 골라 열어 볼 수 있으며 이들 작품에서 어떻게 코드를 짜는가 배울 수 있다.

Saves 는 현재 작업중인 sketch 를 프로세싱의 sketches folder 에 저장한다. 여러분이 지금 작업중인 sketch에 오늘 날자가 아닌 다른 이름을 붙여 주고 싶다면 *File* menu 에 가서 *save As* 로 저장하면 된다.

Exports 버튼은 현재 작업중인 sketch 를 프로세싱의 sketches folder에 내보내는데 이 경우 Java applet 으로 내 보낸다. 이 때 이 applet을 포함하는 HTML file 도 생성된다. 이 기능에 대해서는 나중에 좀 더 자세히 설명할 것이다.

프로세싱 환경에 대한 좀 더 자세한 고급정보를 원한다면 Processing Environment reference.를 참고 하기 바란다.

아래 레벨 미다아 다루기

Director에서는 외부 미디아를 *cast* 에 불러 오거나 새로 만든 다음 이것을 *stage* 에 끌어 오면 *sprite* 가 생성 배치된다. Flash 에서는 외부 미디아를 *library* 에 불러 오거나 새로 만든 다음 stage 로 끌고 와서 *movieclips* 로 인스턴스화 한다. 프로세싱에서는 (자바에서도 마찬가지이지만) 미디아 생성은 모두 코드로 만들어 수행해야 한다. 이것은 HTML 의 작동방법과 같다. 뿐만 아니라 여러분이 창작한 자작 미디아 (벡타그림, DNA데이타, 필름의 색표본, Fargo)를 자바코드의 일부로 포함 시킬 수도 있다. 실제로 어떤 외부 이미지나 사운드도 원한다면 한 파일에 모두 집어 넣을 수 있다. 왜냐하면 이미지의 픽셀도 여러분의 코드의 일부분으로 바꿀 수 있고 사운드 역시 커다란 데이터 배열로 저장할 수 있기 때문이다. library 나 cast 의 이점이란 디스크 공간이나 메모리를 절약하고 형식을 통제하는데 유리하다는 점일 것이다. 또 다른 이점을 들자면 sprite 나 movieclip 은 눈으로 직접 볼 수 있어 버튼, 비디오게임 캐랙터, 독립적인 그래픽스 요소등을 만드는데 실감이 난다는 점이다.그러나 고정된 틀에 의존하는 작업환경은 역기능도 가져 온다. 프로세싱에는 이런 고정된 틀이 없다. 기본적인 drawing routine에 마우스/키보드/시리얼포트의 이벤트 가 다다. 동영상을 만든다는 것은 시간적으로 씬을 다시 바꿔 그린다는 것 뿐이다. 다시 말하면 무비클립이나 스프라이트를 여러분 자신이 써야 한다는 것이다. 그러나 반드시 그럴 이유가 없다. 아티스트에게 유용한 새로운 방법을 발명해 내면 된다. 이렇게 하므로서 매크로미디아 냄새가 풍기는 작품들에서 벗어난 다양화된 심미관을 발견하는 기회가 될 수 있을 것이다.

이 튜토리알의 아래 부분에서는 먼저 이미지를 스크린에 그리는법을 소개하고 시간과 애니메이션을 소개한다. 그런다음 마지막으로 마우스 키보드 그리고 시리알 포트와의 상호작용을 설명할 것이다. 이것들만 알면 높은레벨 툴에서 할 수 있는 어떤 작업도 수행할 수 있다. 여러분 자신이 마음만 먹으면 어떤 작품도 자바로 만드러 낼 수 있다는 사실을 명심하기 바란다.

문법 구조

FlashMX 사용자에게는 별로 새로운 것이 없으므로 복습이라고 생각하고 읽어 주기 바란다. 아래의 보기는 example sketches 에 들어 있는 structure00 이다.

```
// 명령문 과  주석문
// by REAS


// 명령문이란 풀그림을 만드는 요소들이다.
// 명령문 끝에는 ";"를  써 준다.  그것은 명령문의 끝을 의미하는 "statement terminator" 라 부른다.
// 주석문이란 풀그림을 읽는 이의 이해를 돕기 위해 붙이는 주석(설명)을 말한다.
// 주석문은 오른쪽으로 누운 스래시 둘("//") 로 시작한다.


// 만든날 2002년 9월 1일


// size 함수는  윈도의 크기를 얼마로 할 것인가를 컴퓨터에 알려 주는 함수이다.


//모든 함수는 0 또는 그 이상의 인수를 갖는다.
// 인수란 메쏘드에 전달하는 데이터다.  그 데이터는 컴퓨터가 명령을 수행하는데 쓰이는 값이 된다.
size(200, 200);


// background 함수는 컴퓨터에 윈도의 배경색을 지정하는 함수이다.
background(102);
```

그리고 자바 변수들은 :

```
int x = 0;
println(x);
x=x+1;
println(x);
x=x+1;
println(x);
```

실행 버튼을 누른 결과

```
0
1
2
3
```

프래셔들에게: 여기에는 var 가 없다는 점에 유의하기 바란다. 좀더 깊이 알고 싶은 독자는 자바 튜토리알을 찾아서 읽어 보기 바란다. 여기에 변수부분이 있다.

if-then 구문은?

```
int a = 1;
int b = 2;

if(a==b){
  println("same");
}else{
  println("different");
}
```

실행 버튼을 누른 결과

```
different
```

Lingo 와 비교하면 문법은 다소 달라진다. 어떤 값을 대입할 때에 단순 등부호 "="를 사용한다. 이중 등부호 "==" 는 어떤 다른 값과 비교할 때에 사용한다. 즉 "같은가"를 물을 때에 사용한다. 또 "다른가"("not equal to")를 물을 때에는 "<>" 가 아니라 "!=" 를 쓴다. 나머지는 같다. 즉 ("<" , ">" , ">=" , 와 "<=") 는 두 언어에서 차이가 없다. 조건문과 관련된 좀더 자세한 정보는 For more information on conditionals, 전문 문헌을 참조하기 바란다.

반복 실행문(looping)은 ?

```
for(int i=0 ; i<5 ; i++){
   println(i);
}
```

실행 버튼을 누른 결과

```
0
1
2
3
4
```

Lingo 사용자에게 말하면 이것은 "repeat with i = 0 to 4." 와 같다. 소괄호 안에 들어 있는 세 개의 명령문은 두 개의 세미콜론 ";"으로 분리해 놓았다. 첫 번째 명령문은 임시변수를 선언한다. 두 번째 명령문은 반복실행의 조건을 명시한다. 위의 보기에서는 *i* 라는 변수가 5 보다 작은 한 반복실행을 계속하고 그렇지 않게 되면 반복실행을 멈추라는 명령이다. 셋째 명령문에서는 다양한 방식으로 *i* 를 변동시키는 명령을 준다. 위의 보기에서는 한번 반복한 다음 *i* 를 1 증가시키라는 명령을 보여 준다. "i++" 은 "i = i + 1"를 짧게 쓰는 문법규약이다. 썬(SunMicrosytem) 에 가면 반복조건문에 대해서 더 자세한 정보를 얻을 수 있다.

while 조건 반복문은 if-then 조건문과 비슷하다.

```
while(6!=2){
   println("muhuhaha!");
}
```

무한 반복이 계속되니 실행 하지 말것

```
:)
```

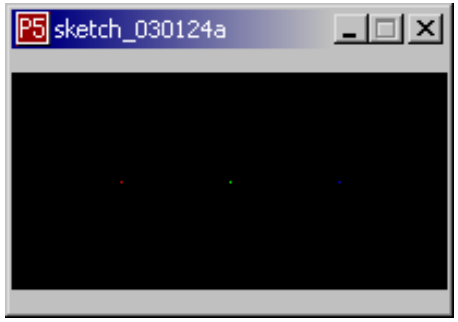
조건분기 흐름 구문(flow-control syntax) 에 관하여 심층 분석에 관심이 있는 독자라면 자바언어의 튜토리알중 조건분기 흐름 구문 부분을 참조하기 바란다.

여기서 완벽한 구문 해설을 할 생각도 할 수도 없지만 흔히 쓰는 루틴(함수)에 대해서 조금 더 설명할까 한다. 단순히 앞으로 집중적으로 사용하게 될 그리기 관련함수들을 초보자에게 이해시킬 목적일 뿐이니 자세한 내용은 Processing Language Comparison 과 Processing Structure Examples.을 참조하기 바란다.

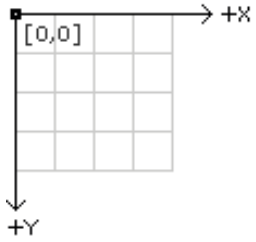
2차원 정적 이미지 그리기

```
size(200,100);
background(0,0,0);
stroke(255,0,0);
point(50,50);
stroke(0,255,0);
point(100,50);
stroke(0,0,255);
point(150,50);
```

이 코드를 실행시키면 오른 쪽과 같은 이미지를 얻을 것이다. 이것은 검은 바탕색의 너비가 긴 윈도우에 빨강, 초록, 파랑색 픽셀 셋이 찍혀 있다.



이제 이 코드를 한줄 한줄 따라가면서 분석해 보자.



첫째로 스크린이란 하나 하나가 (X,Y) 라는 자리표를 갖고 있는 픽셀로 된 그래프라는 점을 명심할 필요가 있다. 이 자리표의 원점은 왼쪽 꼭대기에 놓여 있다. Y가 증가하면 픽셀 위치는 아래로 내려 오고 X가 증가하면 픽셀의 자리는 오른 쪽으로 이동한다. 이것은 보드게임 Battleship에서와 같다. 이 자리표계는 Director 나 Flash에서 쓰는 자리표계와 같다.

```
size(200,100);
```

size함수를 부르는 것은 이 캔버스(윈도우)의 크기를 너비, 200 픽셀, 높이, 100 픽셀로 지정하는 것이다. 이 함수를 처음에 호출하지 않으면 디폴트로 너비 100, 높이 100 으로 지정된다.

```
background(0,0,0);
```

background를 부르는 것은 캔버스 전체의 색을 지정한다. Director에서는 stage 색에 해당되고 Flash에서는 문서(document)의 배경색(background color)이 된다. 인자로 들어간 0,0,0 은 검정색을 뜻한다. 이 함수를 호출하지 않으면 프로세싱은 디폴트로 회색 배경색을 가진 캔버스를 출력한다.

```
stroke(255,0,0);
```

stroke 함수의 호출은 현재 그리기 색깔(stroke color)을 지정한다. 이 함수를 호출한 다음 그리기 관련함수를 호출하면 그림은 모두 이 색깔로 그려진다. 255,0,0 는 빨강을 뜻한다. 이함수를 생략하면 프로세싱은 디폴트로 현재 그리기 색깔을 검정색으로 지정한다.

```
point(50,50);
```

point 함수는 현재 그리기 색깔로 픽셀 하나를 자리표 50,50 에 그리라는 명령이다. 현재 그리기 색깔은 빨강으로 지정되어 있다.

```
stroke(0,255,0);
point(100,50);
```

이 코드는 캔버스 한 가운데에 초록색 픽셀을 하나 찍는다.

```
stroke(0,0,255);
point(150,50);
```

이 코드는 파랑색 픽셀을 오른쪽에 하나 찍는다.

여기서 보듯이 이것은 Lingo 의 이미지를 그리는 draw( ) 함수나 Flash의 ActionScript에 있는 이미지 drawing methods와 비슷하다는 것을 알 수 있을 것이다. 여러분은 스크린을 마치 어떤 그리기 명령을 이해하는 캔버스라고 생각하고 주무르면 된다. 이제 좀 더 복잡한 모양을 그리는 법을 알아 보기로 한다.

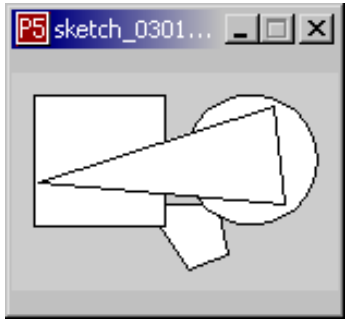
```
background(0,0,0);
stroke(255,255,255);
line(0,0,60,40);
stroke(255,255,0);
line(30,50,100,100);
```

여기서 두 직선을 그려 보기로 한다. 하나는 흰 선이고 두 번째 것은 노랑색 선이다.

이 선그리기 함수 line은 4개의 인수를 받아 들인다. 앞 두 인수는 선분의 시작점 자리표 x,y 이고 뒷부분 두 인수는 선분의 끝 자리표를 나타낸다. 선은 이 시작점에서 끝점까지 그어진다.

다음은 기성 모양을 그리는 명령어들을 보기로 든다.

```
size(150,100);
quad(61,60, 94,60, 99,83, 81,90);
rect(10,10,60,60);
ellipse(80,10,60,60);
triangle(12,50, 120,15, 125,60);
```



triangle 은 세 꼭지점을 가진 세모꼴을 그린다. 이 함수에는 6개의 인수를 받는데 인수 1,2는 첫 꼭지점의 자리표, X,Y 이고 인수 3,4는 둘째 꼭지점, 인수 5,6은 셋째 꼭지점의 자리표, X,Y 이다.

triangle(x1, y1, x2, y2, x3, y3);

quad 는 4 꼭지점을 갖는 다각형을 그린다. 인수의 구조는 세모꼴 명령어 triangle의 인수와 비슷한데 이젠 4번째 꼭지점 자리표, X,Y 가 더해질 뿐이다.

quad(x1, y1, x2, y2, x3, y3, x4, y4);

rect 는 직네모꼴을 그린다. 이 경우 인수는 첫 번째 2개는 위치 자리표 X, Y이고 셋째 넷째 인수는 너비와 높이를 지정한다.

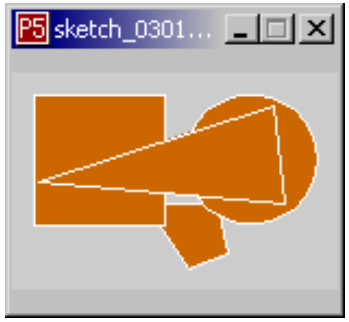
rect(x, y, width, height);

ellipse 는 타원꼴을 그 인수의 구조는 rect 의 인수 구조는 와 같다.

ellipse(x, y, width, height);

이제 이 풀그림을 수정해서 조금 새 기능을 보인다. 아래에 이 새 코드를 마크했다.

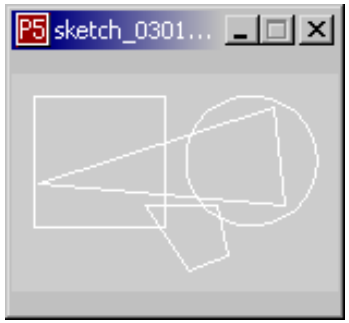
```
size(150,100);
fill(#CC6600);
stroke(#FFFFFF);
quad(61,60, 94,60, 99,83, 81,90);
rect(10,10,60,60);
ellipse(80,10,60,60);
triangle(12,50, 120,15, 125,60);
```



(이번에는 색깔을 조금 다른 기법으로 지정하였다. 이 기법은 HTML 문서의 색깔 지정법과 비슷하다.)

fill 은 stroke 의 사촌과 같다. Fill 은 다각형을 초록으로 칠하고 stroke 은 외각선을 빨강으로 그릴 때에 쓴다. fill 의 인수는 stroke의 인수와 마찬가지로 색깔이 들어 간다. 디폴트의 색깔은 흰색이다. 그런데 모양의 안을 칠하고 싶지 않을 땐 어떻게 할 것인가?

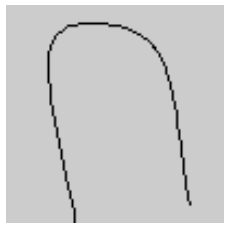
```
size(150,100);
noFill( );
stroke(#FFFFFF);
quad(61,60, 94,60, 99,83, 81,90);
rect(10,10,60,60);
ellipse(80,10,60,60);
triangle(12,50, 120,15, 125,60);
```



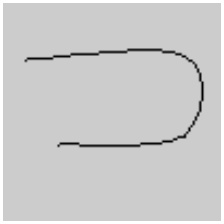
위에서 보듯이 다각형의 안이 칠해지지 않아 타원꼴 밑의 네모꼴 quad 가 보인다. 이것은 색으로 모양의 내부를 채우지 않고 외각선만 그렸기 때문이다. 이와 만찬가지로 noStroke은 외각선을 그리지 않고 다각형의 내부만 채워 칠할 때에 쓴다. 다시 내부를 칠하거나 외각선을 그리고 싶으면 stroke 나 fill 함수를 호출해야만 한다.

곡선을 그리는데에는 직선을 그릴 때보다 약간 복잡하다. 곡선을 지정할 때에는 곡선의 방향과 곡률을 지정해야 하기 때문에 눈에 보이지 않는 비가시 정보를 제공해야 한다. 프로세싱은 curve( ) 와 bezier( ) 함수로 곡선을 그린다.

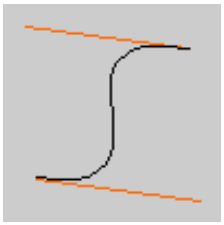
```
curve(84, 91, 68, 19, 21, 17, 32, 100);
```



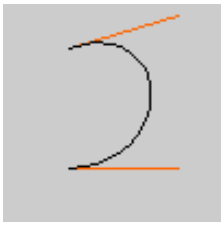
```
curve(10, 26, 83, 24, 83, 61, 25, 65);
```



```
stroke(255, 102, 0);
line(85, 20, 10, 10);
line(90, 90, 15, 80);
stroke(0, 0, 0);
bezier(85, 20, 10, 10, 90, 90, 15, 80);
```



```
stroke(255, 102, 0);
line(30, 20, 80, 5);
line(80, 75, 30, 75);
stroke(0, 0, 0);
bezier(30, 20, 80, 5, 80, 75, 30, 75);
```



curve(x1, y1, x2, y2, x3, y3, x4, y4);
bezier(x1, y1, x2, y2, x3, y3, x4, y4);

curve( ) 함수의 첫째와 둘째 인수는 곡선의 시작점의 자리표를 대입하고 마지막 두 인수는 곡선의 두 번째 점의 자리표를 대입한다. 가운데 인수들은 모양을 정의하는 인수들이다. (이 정의들은 베타버전에 가면 바뀐다- 역자주)

bezier( ) 함수의 첫 두 인수에는 곡선의 첫점의 자리표, X, Y를 대입하고 마지막 두 인수에는 곡선의 끝점의 자리표를 대입한다. 가운데 인수들은 곡선의 모양을 정하는 값들을 대입한다.

위의 bezier( ) 함수 보기에서 보여 주는 오랜지색 직선은 이 곡선의 성질을 규정짓는 숨겨진 조절점을 나타내 준다.

프로세싱에서 이와 같은 모양을 쉽게 그리는 기본 요소를 제공하지만 여러분은 얼마던지 자유로이 여러분 자신의 모양을 만들 수 있다.(또 그러기를 권장한다.)

프로세싱에서는 좀더 복잡한 모양을 그리기 위하여 beginShape( ) 와 endShape( ) 함수를 제공한다. beginShape( ) 는 어떤 모양의 꼭지점을 받아 쓰기 시작하고 endShape( ) 에서 받아 쓰기를 멈춘다. The beginShape( ) 명령은 인수를 받아 들이는데 이 인수는 꼭지점 데이터로 어떤 모양을 그릴 것인가를 지정한다. beginShape( ) 안에 들어 가는 상수는 LINES, LINE_STRIP, LINE_LOOP, TRIANGLES, TRIANGLE_STRIP, QUADS, QUAD_STRIP과 POLYGON 이 모두다. beginShape( ) 명령 다음에는 vertex( ) 명령이 따르는데 모양을 그리기에 충분한 데이터가 입력되면 endShape( ) 명령을 반드시 써 넣어야 한다. vertex( ) 함수에는 2차원 2D그림을 그리는 경우에는 X, Y 라는 두 개의 꼭지점 자리표를 인수로 받아 들이고 3차원 3D그림을 그리기 위해서는 3차원 공간의 자리표인 X,Y,Z 라는 3개의 실수를 인자로 받아 들인다. 이 모양들은 현재 stroke 색깔로 외각선을 그리고 현재 fill 색깔로 내면을 채워 색칠한다. (좀더 자세한 정보는 나중에 나오는 color 부분을 참조할 것).

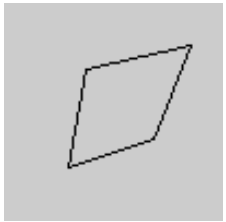
아래와 같은 주석문이 프로세싱 참고문헌에 나와 있다.

프로세싱은 뾰족 다각형만 그릴 수 있다. 그러나 오목다각형도 그릴 수 있도록 현재 작업중이다. 수정된 새 버전에서는 인자로 CONVEX_POLYGON 과 CONCAVE_POLYGON을 쓸 것이다. (이 예고와는 달리 개정된 베타 버전에서 오목다각형도 구별 없이 공통인수 POLYGON를 쓴다 -역자주)

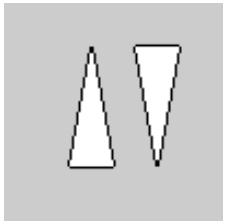
그렇다 하더라도 뾰죽다각형만으로도 얼마던지 다양한 모양을 그릴 수 있다. 웹사이트를 검색하면 좋은 보기들을 찾을 수 있을 것이다.

아래에 Proce55ing.net에서 찾은 보기들을 올려 놓았다.

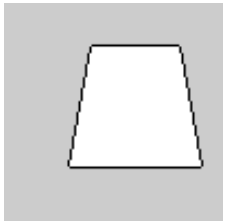
```
beginShape(LINE_LOOP);
vertex(30, 20, -50);
vertex(85, 20, 0);
vertex(85, 75, -80);
vertex(30, 75, 0);
endShape( );
```



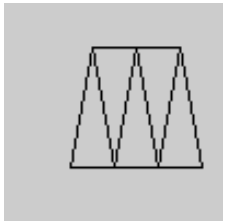
```
beginShape(TRIANGLES);
vertex(30, 75);
vertex(40, 20);
vertex(50, 75);
vertex(60, 20);
vertex(70, 75);
vertex(80, 20);
vertex(90, 75);
endShape( );
```



```
beginShape(TRIANGLE_STRIP);
vertex(30, 75);
vertex(40, 20);
vertex(50, 75);
vertex(60, 20);
vertex(70, 75);
vertex(80, 20);
vertex(90, 75);
endShape( );
```

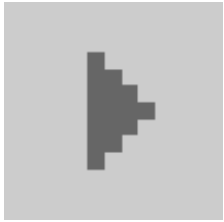


```
noFill( );
beginShape(TRIANGLE_STRIP);
vertex(30, 75);
vertex(40, 20);
vertex(50, 75);
vertex(60, 20);
vertex(70, 75);
vertex(80, 20);
vertex(90, 75);
endShape( );
```

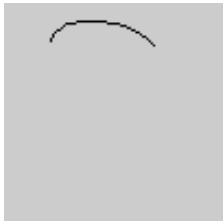


```
noStroke( );
fill(153, 153, 153);
beginShape(TRIANGLE_STRIP);
vertex(30, 75);
vertex(40, 20);
vertex(50, 75);
vertex(60, 20);
vertex(70, 75);
vertex(80, 20);
vertex(90, 75);
endShape( );
```

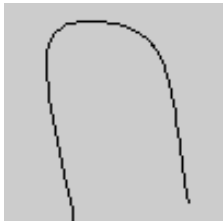
```
noStroke( );
fill(102);
beginShape(POLYGON);
vertex(38, 23);
vertex(46, 23);
vertex(46, 31);
vertex(54, 31);
vertex(54, 38);
vertex(61, 38);
vertex(61, 46);
vertex(69, 46);
vertex(69, 54);
vertex(61, 54);
vertex(61, 61);
vertex(54, 61);
vertex(54, 69);
vertex(46, 69);
vertex(46, 77);
vertex(38, 77);
endShape( );
```



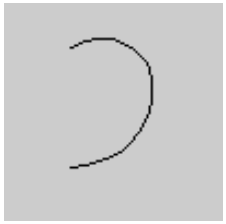
```
beginShape(LINE_STRIP);
curveVertex(84, 91);
curveVertex(68, 19);
curveVertex(21, 17);
curveVertex(32, 100);
endShape( );
```



```
beginShape(LINE_STRIP);
curveVertex(84, 91);
curveVertex(84, 91);
curveVertex(68, 19);
curveVertex(21, 17);
curveVertex(32, 100);
curveVertex(32, 100);
endShape( );
```



```
beginShape(LINE_STRIP);
bezierVertex(30, 20);
bezierVertex(80, 0);
bezierVertex(80, 75);
bezierVertex(30, 75);
endShape( );
```



벡타 그리기에 대한 더 자세한 정보를 얻고자 한다면 Processing Form Examples, 이나 Processing Shape reference를 방문하기 바란다.

스크린에 그리거나 나타내게 하는 방법은 더 많이 있지만 2차원 그림에만 주력하였다. 이것만으로도 애니메이션과 유저와의 서로작용을 설명할 수 있기 때문에 일단 그리기 기법의 기초는 접고 시간과 동영상 기법에 대한 설명으로 들어간다. 그런 다음에 다시 그리기 기법에 돌아 올 것이다.

시간과 동영상

Director에는 score 가 있다. playback head 와 sprite를 위한 tweening methods 가 있다. 비디오라든가 내포된 플래시, QTVR, 사운드 등은 자신의 시간 스페측에서 애니메이트 한다. 동적인 에니메이션을 가미하고 싶으며 프래임하나를 사용하여 ExitFrame 또는 PrepareFrame 이벤트를 사용하여 여기에 코드를 첨가한다. 플래시에서는 타임라인이 있고 조금더 Director 보다 정교한 트윈닝을 지원한다. 액션스크립트만을 사용하는 이는 두 개의 프래임만을 사용하기도 한다. 하나에는 에니메이션을 준비하는 setup에 여기서 루틴을 호출한다. 또 다른 프래임에서는 무한 루프의 코드를 작성한다. 또한 액션스크립트에는 onClipEvent (enterFrame)에 대응하여 작동하는 코드를 적어 넣어 애니메이션을 하기도 한다. 프로세싱에는 타임라인이라 스코어라는 것이 없다. 그러나 Lingo 나 Actionscript와 마찬가지로 프로세싱에서는 여러분 자신의 그리기 루틴을 접목시킨 프래임진행 event handler를 쓸 수 있다. 이제까지는 정적 그림을 그리는데에만 적용되는 프로세싱의 Basic Mode만 보여 주었다. 프로세싱을 돌리는 데에는 Basic Mode를 포함하여 세가지 모드가 있다. 나머지 둘은 Standard Mode. 와 Advanced Mode들이다. Advanced Mode는 훈련용 바퀴가 없는 진짜 자바다. 시간과 움직임을 풀그림하기 위해서는 최소한 Standard Mode를 써야 한다. 이 튜토리알에 필자가 간간히 집어 넣은 링크를 따라 갔던 독자라면 standart mode 로 짠 풀그림을 한 둘 보았을 것이다. 아래에 이 모드로 짠 간단한 풀그림 하나를 보기로 들기로 하자.

```
int x = 0;

void setup( ){
   noStroke( );
}

void loop( ){
   background(190);
   rect(x, 0, 5, 100);
   x=x+1;
}
```

이 보기에서는 흰색 직네모꼴이 왼쪽에서 오른쪽으로 단 한번 움직인다.

오른쪽에 애니메이트 GIF를 보여 주고 있다.

standard 는 이 setup( ) 이라는 함수가 포함되는데 이 함수는 풀그림이 실행하면 단 한번 돈다. 다음의 loop( ) 함수는 그 뒤에 따르는 중괄호안의 코드를 무한회수 반복 실행시킨다. Lingo에서 setup 함수는 beginSprite 나 startMovie 와 같은 것이고 loop( ) 함수는 ExitFrame 이나 PrepareFrame에 해당된다. Flash에서는 이 setup( ) 함수가 첫프래임으로 단 한번 실행하고 loop( ) 함수를 호출하는 것과 같다. 여러분은 이 함수들을 조직화하고 캡슐화 하기 위해 스스로 작성할 수도 있다. 자작함수를 쓰는 법에 대해서 좀더 알고 싶다면 Sun의 자바 강좌 – Implementing Methods section.부분을 참조하기 바란다.

일단 함수를 하나 써서 쓰기 시작하면 프로세싱은 자동적으로 표준 모드(standard mode)로 바뀐다. 그런 다음에는 프로세싱은 함수 바깥에 있는 어떤 명령도 인식하지 못한다. 함수 바깥에 쓸 수 있는 것은 변수를 선언하고 초기화하는 것 뿐이다. 그러므로 초급 모드(basic mode)에서 사용했던 명령들은 모두 setup( ) 이나 loop( )함수 안에 집어 끌어 다 넣어야만 한다. 전역변수 (즉 함수 밖에서도 살아 있는 변수)만이 setup( ) 이나 loop( )함수 밖에서 선언되고 초기화된다. 위의 보기에서 x 가 그런 전역 변수가 된다.

프로세싱에는 framerate(n) 가 있는데 이 함수를 써서 스켓치 전체의 속도를 조절한다. 그러나 스켓치의 일부를 다른 속도로 움직이게 할 수 있다. 이것은 그 부분의 변화(움직임)의 증가량을 늘려 주거나 실수(float)를 써서 변화량을 분수량으로 만

들어 줄여 주면 된다. 또 긴 시간을 쓸 때에나 정밀한 시간 조절을 위해서는 프로세싱은 아래와 같은 시간 함수를 제공한다. 아래 함수는 모두 컴퓨터의 시간을 불러 오는 함수들이다.

year( ) // 올해 값을 반환한다. 즉, 2002, 2003, 따위를 반환한다.
month( ) // 이번 달을 반환한다. 즉 1에서 12까지의 수를 반환한다.
day( ) // 오늘 날짜를 반환한다. 즉 1에서 31까지의 수를 반환한다.
hour( ) // 현재 시간 즉, 0에서 23까지의 수자를 반환한다.
minute( ) // 현재 분을 반환한다. 0에서 59까지의 수를 반환한다.
second( ) // 현재 초를 반환한다. 0에서 59까지의 수를 반환한다.

그밖에 특수한 함수 millis( ) 가 있는데 이것은 애플렛이 시작한 (풀그림이 실행하기 시작한) 이후의 경과 시간을 1000 분의 1초 단위로 잰 값을 반환한다. 이 함수는 애니메이션 경과시간을 조절할 때 쓸 수 있다.

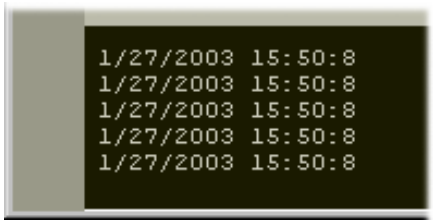
millis( ) // 애플렛이 시작한 이후 경과시간을 1000분의 1초 단위로 반환한다.

그리고 애플렛을 지연시키는 (기다리게 하는) 함수가 있다. 이 함수를 잘 쓰면 이 역시 프래임속도를 조절하는데에 쓸 수 있다.

delay(40); // 는 애플렛을 40 milliseconds(1000분의1초) 기다리게 하는 명령이다.

```
void loop( ){
    print(month( )+"/");
    print(day( )+"/");
    print(year( )+" ");
    print(hour( )+":");
    print(minute( )+":");
    println(second( ));
}
```

이것은 올해 년수, 오늘 날짜와 현재 시간, 현재 분, 초를 프로세싱 아래의 문자 출력창에 계속해서 업데이트 해가면서 출력하게 하는 애플렛이다.



위의 보기는 썩 좋은 보기라고는 할 수 없지만 시간 함수를 간단하게 이해시키기에 충분하다고 본다. 좀더 근사하고 복잡한 시간 함수의 보기를 원한다면 Clock, by Mescobosa, 와 Milliseconds, by REAS를 참조하기 바란다. 이 두 참고자료는 모두 프로세싱으로 작성된 것들이다. 에니메이션에 대한 참고 자료로는 Processing Motion Examples를 찾아가 보기를 바란다.

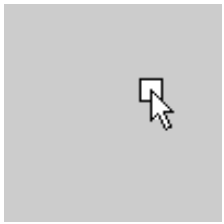
마우스와 키보드

마우스와 키보드 사용법은 Flash 나 Director의 사용법이나 비슷하다. Lingo 에서는 the mouseLoc, the mouseH, the mouseV 그리고 mouseDown따위가 있다. 또 Flash에는 onClipEvent (mouseDown)를 많이 쓴다. 프로세싱에서는 mousePressed( ) 가 마우스를 누를 때 마다 호출된다. 또 mouseReleased( ) 함수가 마우스를 눌른 상태에서 뗄 때 마다 호출된다. 여기에 원하는 액션에 대한 코드를 적어 넣으면 된다. 이것은 loop( ) 다음의 중괄호에 코드를 적어 넣는 것과 마찬가지이다.

```
void loop( ) {
    background(190);
    rect(mouseX-5, mouseY-5, 10, 10);
}

void mousePressed( ) {
    fill(0);
}
void mouseReleased( ) {
    fill(255);
}
```

이 보기는 마우스를 따라 다니는 작은 네모꼴을 보여 준다. 마우스를 누르면 정네모꼴은 검정색으로 변한다.

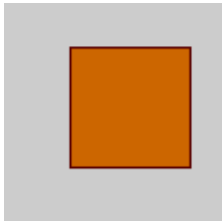


마우스에 대한 자세한 정보는 Processing Mouse reference,를 참조하고 또 exquisite Processing Mouse examples도 찾아 가 보기 바란다.

키보드 입력 역시 Flash 와 Director와 비슷하다.

```
void loop( ) {
   if(keyPressed) {
      fill(102, 0, 0);
   } else {
      fill(204, 102, 0);
   }
   rect(30, 20, 55, 55);
}
```

이 보기는 키보드의 아무 키나 누르고 있으면 정네모꼴의 색깔이 어두운 빨강색으로 바뀐다. 이 경우 배경색을 다시 그릴 필요가 없다!



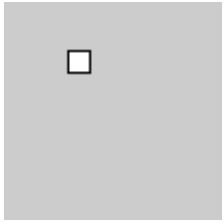
키보드 입력은 이벤트 다루기함수 형식으로 사용할 수 있다. keyboard input can also be delivered to you in the form of an event handling function.

```
int x = 50;
int y = 50;

void loop( ){
   background(190);
   rect(x,y,10,10);
}

void keyPressed( ){
   if(key=='w'||key=='W'){
      y--;
   }else if(key=='s'||key=='S'){
      y++;
   }else if(key=='a'||key=='A'){
      x--;
   }else if(key=='d'||key=='D'){
      x++;
   }
}
```

이 보기에서는 키보드의 특정 키를 눌러 네모꼴을 움직이게 한다.



키보드에 더 자세한 정보는 Processing Keyboard reference, 를 참고 하고 특이한 용법을 보고 싶으면 exquisite Processing Keyboard examples도 방문하기 바란다.

프레센테이션/내보내기

IFlash나 Director에서 보면 키보드나 메뉴 항목을 써서 풀그림을 전체 스크린에서 작동하게 할 수 있다. 풀스크린 모드는 인스톨할 때라던가 프레센테이션을 할 때 매우 편리하다. 프로세싱에도 이런 기능이 있다. 즉 메뉴에서 Sketch > Present, 를 선택하거나 Ctrl+P (⌘+P on a Mac)을 하면 풀스크린 모드가 된다. 또 SHIFT 키를 누른 상태에서 실행 버튼을 누르면 풀스크린 모드로 여러분이 작성한 코드가 한 가운데에서 실행되는 것을 볼 수 있다. 풀스크린 모드에서는 배경색은 오른쪽 그림과 같은 어두운 회색이 되고 ESC 키나 왼쪽 아래에 있는 "stop" 버튼을 클릭하면 정상모드로 되돌아 온다.

여러분이 작성한 어떤 프로세싱 풀그림도 자바 애플렛을 만들어 웹페이지에 끼워 넣어 "발표" 할 수 있다. 메뉴에서 File -> Export to Web 하거나 단축키 Ctrl+E 를 눌러도 된다. (export 단추를 눌러도 된다.) 어느 단추인지 모른다면 모든 단추에 마우스를 갖다 대면 풍선도움말 "Export" 가 나타나는 단추를 찾을 수 있다. 이 단추를 눌러 내보내기를 실행하면 "Exporting for web . . ." 라는 메시지를 스테터스 창에서 보게 되고 내보내기가 끝나면 "Done Exporting." 라는 메시지가 나타난다. 그러면 애플랫은 어디에 있을까? Processing 의 sketch folder를 열어 보면 applet라는 디렉토리가 생성된 것을 발견할 수 있다. 이 폴더를 그냥 올려놓기(upload)하면 된다. 그러나 이 폴더에 생성된 디폴트 index.html을 편집하여 올려 놓기를 권장한다. 그러나 그곳에 있는 다른 모든 파일들의 상대경로는 그대로 유지하여야 한다. 여기서의 규칙은 일반적인 HTML 미디아 파일의 규칙과 마찬가지이므로 그대로 따르면 된다.

아래에 내보내기 기능이 만든 파일들과 그 상대경로를 표시하였다.

```
Processing Folder/
        sketchbook/
            default/
                your_sketch_name/
                                applet/
                            your_sketch_name.java
                            your_sketch_name.class
                            your_sketch_name.jar
                            index.html
```

save( ) 와 saveFrame( )
양방향성이 아닌 형식을 내보내기 할 때에는 프로세싱 윈도우를 tif 파일 형식으로 내 보낼 수 있다. 이 경우 saveFrame( ) 함수를 쓰면 된다. 이 함수는 loop( ) 함수안의 코드 끝에 첨가한다. saveFrame( ) 이 여러번 호출되면 screen-0001, screen-0002, screen-0003, 의 순서로 프래임이 저장된다. save( ) 함수는 현 윈도우에 존재하는 이미지를 저장한다. 인수에 파일 이름을 적어 넣어 주면 그 파일이름으로 그림이 저장된다. 이런 일련의 연속 이미지를 쓰면 Quicktime 이나 다른 비디오 풀그림을 써서 이미지를 순차적으로 보여 주므로서 동영상 효과를 얻을 수 있다. 이처럼 프로세싱은 쉽게 이미지를 저장할 수 있는 내장함수가 있지만 노력을 좀 더 들여서 풀그림을 짜면 다른 포맷으로 이미지를 내 보낼 수도 있다. 여기에 Adobe Illustrator에 내보내기한 보기하나를 소개한다.

이미지 파일 그리기

프로세싱에서 이미지를 스켓치에 올려 놓기는 쉽다. 자바는 JPG 또는 GIF 형식의 그림 파일만 받아 들인다. (좀더 작업을 하면 다른 그림도 올릴 수 있겠지만.) 그림 파일 시스템을 만들고 코드 두어줄 써주면 된다. 먼저 스켓치를 저장한다. 그런 다음 저장된 스켓치가 있는 폴더에 가서 폴더를 열어 보면 data 라는 하위 폴더가 생성되어 있는 것을 확인할 수 있다. 그 폴더에 불러다 쓸 그림 파일을 저장한다. 아래에 디렉토리 구조를 보여 준다.

```
Processing Folder/
        sketchbook/
            default/
                여러분이 만들어 저장한 스켓치 폴더/
                                data/
                                    불러다 쓸그림.gif
```

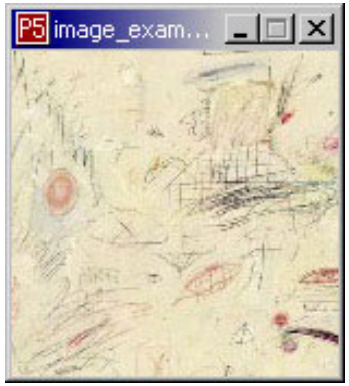
보기를 들자면 내가 image_example_1 라는 스켓치를 만들어 Cy Twombly가 그린 아래와 같은 이미지 twombly.jpg를 올려 놓을 생각이라면 이그림 파일을

아래와 같은 폴더구조를 갖도록 배치하여야 한다.에

Processing Folder/
sketchbook/
default/
image_example_1/
data/
twombly.jpg

그런 연후에 아래와 같은 코드를 적어 넣는다.

```
size(150,150);
BImage b = loadImage("twombly.jpg");
image(b,0,0,150,150);
```

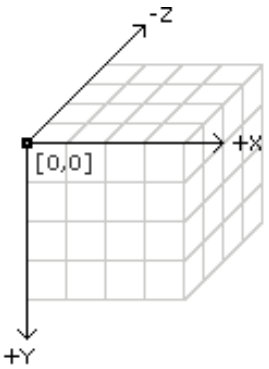


BImage 란 이제 불러다 스크린에 그릴 그림을 담을 객체다. b 는 이 개체의 이름(인스턴스명)이다. image( ) 는 바로 이 그림을 스크린에 그리라는 명령이다. 그 구문 형식은 아래와 같다.
image(BImage, x, y, 너비, 높이);
위에서 너비와 높이는 생략할 수도 있다. 그러면 그림은 제 크기로 그린다. l

데이터 폴더에 있는 그림 말고 URL 주소를 가진 그림도 올릴 수 있다.

이미지에 관하여 자세한정보를얻고 싶으면 프로세싱의 참고문헌을 참조하기 바란다. 여기에 그 부분 Loading_and_Displaying이 있다. 또 이 지식에 기반을 두고 순차적 이미지 (video footage)를 올릴 수 도 있다. 좀 더 재미있는 보기를 원한다면 Processing Image Examples를 보기 바란다.

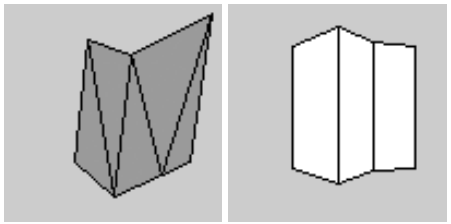
3D 형태



내재적으로 2차원 환경에 3차원을 도입한다는 것이 과연 타당하냐 하는 논란은 많았다. Flash 의 경우 제3자 벤더들이 각가지 3차원 도구를 제공하고 있다. Director의 경우에는 최근에야 3차원 벡타 그래픽스 스프라이트를 덧 입히듯 도입하였다. 그러나 그 사용법이 너무 복잡하여 대부분의 유저들은 감히 배울 엄두조차 내지 못하는 형편이다.

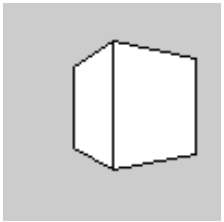
프로세싱에서 3차원이란 단순히 z-축 하나를 더 도입하는 것 뿐이다.

```
vertex(x, y, z);
line(x1, y1, z1, x2, y2, z2);
bezierVertex(x, y, z);
curveVertex(x, y, z);
```

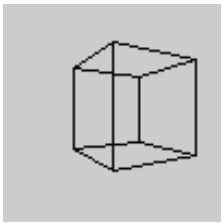


```
box(size);
box(width, height, depth);
sphere(size);
```

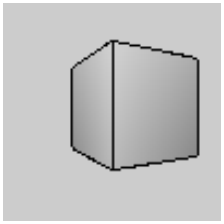
```
translate(58, 48, 0);
rotateY(0.5);
box(40);
```



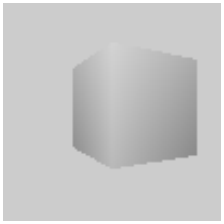
```
noFill( );
translate(58, 48, 0);
rotateY(0.5);
box(40);
```



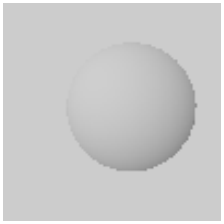
```
lights();
translate(58, 48, 0);
rotateY(0.5);
box(40);
```



```
noStroke();
lights();
translate(58, 48, 0);
rotateY(0.5);
box(40);
```



```
noStroke();
lights();
translate(58, 48, 0);
sphere(28);
```



여기서 주목할 것은 box 나 sphere 는 위치의 자리표를 묻지 않는다는 점이다. 그 대신 translate 와 rotate를 써서 입체의 위치 자리표와 그 지향을 지정한다. 그리고 scale 과 push 와 pop 이라는 한쌍의 명령어를 써서 병진운동과 회전운동을 잘 조직적으로 갈무리하고 불러 오고 한다. 3차원 그리기를 어떻게 조직화하고 갈무리 하고 불러 오고 하는 기법을 배우려면 Processing Transform Reference 와 Processing Transform Examples를 방문하기를 바란다. 물론 이런 변환에 관심이 없다면 이런 방법이 있다.

또 주목할 점은 lights() 와 noLights()용법이다. 조명을 사용하면 3차원 물체에 명암을 나타낼 수 있다. 3차원 입체에 대한 조명에 관하여 배우고 싶으면 Processing Lights Reference.를 참조하기 바란다.

"뭐 그게 3차원의 전부야?"

아마도 여러분은 이것으로는 성에 차지 않을지 모른다. 그렇다면 이것만으로도 얼마나 훌륭한 작품을 만들 수 있는지 직접 아래에 가서 프로세싱만으로 마든 무른모들을 둘러 보기 바란다. Processing Software. 그리고 이것은 오진 시작일 뿐이라는 점에 유의하기 바란다.

픽셀

플래시에서 픽셀을 제어하기는 아직도 요원하다. 디렉터에는 이제야 SetPixel 과 GetPixel 를 도입했다. (그래서 디렉터를 좋아 하는 한 유명한 양방향성 미디아 아티스트는 SetPixel이라 니크네임을 쓰기까지 한다.) 그러나 Director는 아마도 픽셀을 다루는 시시템으로는 가장 느린 무른모일지 모른다. ITP의 Danny Rozin 교수가 가르치는 The World – Pixel by Pixel 이라는 강의를 수강하는 학생은 거의 모두가 C 로 풀그림을 하는데 C가 유일하게 자신들의 컨셉 목적을 달성케 해주는 빠른 속도를 보장하기 때문이다. (Lingo 와 MAX 만이 유일한 대안이 되고 있다.) 그래서 그 과목을 듣는 학생들은 C 강의를 듣는 것이 유행이 되다 싶이 되었다. 프로세싱의 픽셀작업은 Lingo 보다 훨씬 빠르며 말할 것도 없이 훨씬 간단하다. 자바의 픽셀 작업은 C 의 픽셀보다는 느리다는 것은 사실이지만 프로세싱은 C 보다 배우기 쉽기 때문에 Danny Rozin 의 강의를 듣는 학생들은 프로세싱을 쓰기 시작하지 않을까 예상하고 있다.

```
get(x, y); // 정수를 반환한다.
set(x, y, color);
pixels[index]; // 출력 화면의 픽셀 배열
```

```
int width = 100;
int height = 100;
BImage b; // declare variable "b" of type BImage
b = loadImage("basel.gif");

image(b, 0, 0);
for (int i=30; i<(width-15); i++) {
    for(int j=20; j<(height-25); j++) {
        color here = get(30, j);
        set(i, j, here);
    }
}
```

이와 같은 픽셀 제어 기능을 갖게 되면 여러분 자신의 그리기 루틴을 만들어 낼 수도 있다. 여기에 투명도를 제어하는 풀그림 보기가 있다. 프로세싱으로 Director의 ink 대부분을 만들어 낸다는 것도 그리 어려운 일은 아니다. 여기에 점선 그리기 함수의 보기가 있다.

픽셀 다루기에 대해 더 자세한 정보를 원한다면 Processing Image reference 와 Processing Image Examples.를 참조 바란다.

문자 출력

현재 프로세싱에서는 프로세싱 특유의 폰트 형식을 써서 문자 출력 시시템을 구축하고 있다. 프로세싱 제작자들은 끌어오기(import) 메뉴 아이템을 탑재하여 상당히 다양한 폰트를 쓸 수 있게 배려하였다. 현재 프로세싱에서 쓸 수 있는 폰트 타입을 보려면 여기를 클릭 해 보기 바란다. 아래에 문자출력을 포함하는 간단한 풀그림을 보기로 내 놓았다. 이 풀그림을 복사해서 붙여 넣기 한 다음 풀그림을 실행시켜 보기 바란다. 아마도 에러가 나올 것이다.

```
size(200,100);
background(#FFFFFF);
fill(#000000);
BFont f = loadFont("Bodoni-Italic.vlw.gz");
textFont(f, 50);
text("handglove", 14, 60);
```

에러 메시지는 Bodoni-Italic.vlw.gz 찾을 수 없다는 오류메시지일 것이다. 그 이유는 지금 불러 쓸 폰트를 data 폴더에 끌어

오기를 하지 않았기 때문이다. (데이터 폴더가 무엇을 뜻하는지 모르는 독자는 이 튜토리알의 이미지 출력 부분을 다시 한번 읽기 바란다.) This is because you have not yet imported the font file into your data folder. (see the images section of this tutorial for more information on your data folder). 먼저 원하는 글자체를 선택 한 다음 프로세싱 폴더에 있는 fonts 폴더에 가서 골라 놓은 폰트를 복사해다 스켓치 폴더의 data 폴더에 복사해 놓는다. 그런 다음 풀그림을 돌리면 에러 메시지 없이 문자가 잘 출력될 것이다.

BFont f = loadFont("Bodoni-Italic.vlw.gz"); 폰트 파일을 변수 f 에 장진한다.

textFont(f, size); 문자를 그리기 전에 먼저 현재 그릴 폰트와 그 크기를 지정한다.

text("handglove", x, y); 지정한 자리에 문자를 출력한다.

이 보기는 회전과 간단한 for 반복문을 결합한 코드다.

```
size(200,100);
noStroke( );
BFont f = loadFont("Univers66.vlw.gz");
textFont(f, 50);
fill(#FFFFFF);
ellipse(-50,-55,150,150);
fill(#CC6600);
for(int i=0;i<20;i++) {
   rotateZ(0.2);
   text("dizzy", 90,0);
}
```

여러분이 문자 출력에 대단한 취미가 없다면 간단한 방법 이 있다. 또 문자 입력 필드가 필요한 독자라면 자신만을 위한 루틴을 작성하는 것도 유용할 것이다. 자신만의 특유한 유저 인터페이스 도구들을 프로젝트에 첨가하는 것은 그리 어려운 일이 아니다. 그것을 처음 부터 시스템이 제공하는 고유의 인터페이스라 생각하지 말고 자신의 스타일에 맞는 습작 모듈이라 생각한다면 그리 큰 부담을 느끼지 않을 것이다. 그렇게 함으로서 다른 풀그림의 탑재 컴포넌트를 쓰는 것 보다 디자인상의 자유를 만끽할 수 있는 이점이 있다. 자신의 특유한 인터페이스를 고안하려는 생각이 있는 독자라면 Processing GUI Examples을 참조 바란다.

또 별난 프로세싱 문자 출력 보기도 있다. 프로세싱은 심미학과 전자계산에 관심이 많은 이들이 모인 그룹에서 출발한 것이다. 그들이 추구하는 문자출력의 새로운 형식도 MIT Media Lab research group 을 세계적으로 유명하게 만든 계기가 되었다는 것을 알아 두기 바란다.

시리알 장치

시리알 장치를 통해서 컴퓨터는 Palm Pilot나 의료장치와 같은 어떤 외부 기기와도 통신을 할 수 있다. 전자 아트세계에서는 시리알 포트를 통해서 자작 외부 기기와 통신을 하는 것이 보통이다. Director에서는 제3자 제품인 Xtra를 통해서 외부장치와 통신할 수 있다. Flash 는 아직 외부 장치와 통신할 수 없으며 제3자 plugin system도 나와 있지 않다. 프로세싱에서는 시리알 포트를 통한 통신 시스템이 내장되어 있다. 여기에서는 돌아가는 외부 손잡이와 프로세싱 스켓치와 서로 작용하는 보기 하나를 들어 간단한 설명을 할까 한다. 이야기에 들어 가기전에 이 마디는 앞 마디들 보다 기술적으로 어렵다는 사실을 미리 경고해 두고자 한다. 전기회로에 대한 지식이 좀 있어야 이해할 수 있기 때문이다.

ITP에서는 BX-24 라는 집적회로 카드를 쓴다. 다른 비슷한 곳에서도 이 장치를 쓰고 있다. BX-24 장치에 대해 더 자세한 정보를 원한다면 Tom Igoe's Physical Computing reference 나 ITP의 Tom Igoe교수가 하는 Physical Computing 강좌의 실험실 과제물 을 참조하기 바란다.

왼편 사진이 이 회로 장치이다. 사진을 간명하게 하기 위해 빼어 버린 +5v짜리 전원장치가 더 있다. 이 장치는 10K 짜리 전위차재개(potentiometer) 와 13째 핀에 1K 짜리 저항을 쓰고 있다. 좀 더 자세한 정보는 ITP Intro to BX-24를 참조하기 바란다.

```
sub main()
   delay 0.5
   do
      debug.print cStr(getADC(13))
      delay 0.1
   loop
end sub
```

이것이 칩에 내려받기 하는 풀그림이다. 이것은 손잡이의 각도를 컴퓨터에 되돌려 주는 일밖에 하지 않는다. 이 풀그림의 작동여부는 BasicX debug 창에 수자가 연속해서 흘러 내려가는 것을 보면 알 수 있다.

프로세싱은 메뉴 Sketch -> Serial Port를 통해서 어느 포트를 통신포트로 사용할 것인가를 지정해 줄 수 있다.

```
String buff = "";
int val = 0;

void setup() {
   beginSerial(19200);
}

void loop() {
   background(val,val,val);
}

void serialEvent() {
   if(serial!=10){
      buff += (char)serial;
   }else{
      buff = buff.substring(0,buff.length()-1);
      val = Integer.parseInt(buff)/4;
      buff = "";
   }
}
```

이 프로세싱 풀그림은 외부 장치의 손잡이가 돌아 감에 따라 배경색이 검정에서 흰색으로 바뀌도록 한 것이다.

좀더 자세한 정보는 Processing serial reference를 참조 하기 바란다.

전망

프로세싱은 현재 진행형인 따끈따끈한 최신 프로젝트다. 앞에서도 말했지만 계속 웹사이트를 방문하여 새 업데이트가 없는지 살펴 보아야 한다. 새 버전은 낡은 버전자리에 쉽게 바꿔 놓을 수 있으며 sketchbook/default 는 복사하여다 새 버전의 자리에 옮겨 놓으면 된다. 이 문서를 처음 쓸 때에는 ALPHA 0050 이던 것이 2004년 가을에 돌아와 이 튜토리알을 업데이트하려다 보니 ALPHA 0068 로 업데이트되어 있었다. 프로세싱은 기존의 3 모드의 이름인 basic, standard와 advanced 란 이름도 바꿀지 모른다. 또 Casey 가 말하기를 Proce55ing 이란 이름도 Processing, 이라고 바꿀지도 모른다 하였는데 그런 변화들이 실현되었다. 그래픽스엔진은 anti-aliased 도입하여 크게 향상되었고 색체에 알파값을 도입하여 투명도를 조절할 수 있게 하였다. 물론 기존의 시스템에도 약간 연구하면 이런 것들을 할 수 있었지만 이젠 손쉽게 구현할 수 있게 됐다. 프로세싱은 그 자체가 프로세싱을 풀그림하는 언어로 쓴 것이다. 이것은 Flash 나 Director 와 다른 점이다. 거기서는 대부분 C로 쓴 것이다. 또 Amit Pitaru 는 Sonia를 만들어 사운드를 다루는 라이브러리를 제공하였다. 네트워크 통신이 매우 중요하므로 앞으로 할 일 목록에 가장 앞에 나와 있다. 이것은 웹에서 파일을 내려받을뿐 아니라 telent, FTP, Gnutella, Carnivore, 등과 통신하고 기타 응용 풀그림, Flash 나 Director 와 같은 프로그램과도 연동할 수 있는 통신기능을 지원하게 하려 하고 있다. 프로세싱의 포럼이나 Discussion사이트에는 이런 풀그림들이 하나 둘 올라 오고 있으며 머지 않아 프로세싱은 이런 기능들을 공식적으로 지원하게 될 것이다. Camera vision에 관해서는 JMyron라는 라이브러리가 있다. 이것은 Director 용 오픈소스 camera vision Xtra를 프로세싱에 접목 구현한 것이다. 프로세싱도 통합개발환경을 구축한다는 정신으로 색상표나 베지어 편집기 따위를 내장도구로 제공할 생각을 갖고 있다. 프로세싱의 저자인 Ben 과 Casey 는 무른모의 개선뿐 아니라 프로세싱 커뮤니티의 확장도 계획하고 있다. 유용한 코드조각들을 모아 두는 저장 창고도 계획하고 있다. 프로세싱은 오픈 소스이다. 이말은 누구나 프로세싱 자체를 다시 편집할 수 있다는 것을 뜻한다. 적절한 오픈소스 프래임워크를 구축하여 누구나 소스를 내려 받아 다시 컴파일하여 새 버전을 내어 놓는데 이바지 할 수 있게 할 것이라 한다. 이미 상당량의 프로세싱 확장 라이브러리가 축적되어 있다.

이 모든 것을 생각한다면 매크로미디아 제품을 배우고 있거나 이미 통달한 이에게 프로세싱은 유용한 도구가 될 것이라 생각한

다. 여러분이 만든 프로세싱 작품들을 우리에게 보내주거나 공개해서 우리들도 여러분이 탐험한 결과물을 배우게 해 주기 바란다. 또 온라인 커뮤니티에 들어와 토론장에서 함께 경험을 나누기를 바란다. 여러분의 즐거운 창작생활을 기원하며.

Josh Nimoy 는 2004년 뉴욕대학(New York University)의 Tisch School of the Arts 에 속한 Interactive Telecommunications Program (ITP)의 대학원생이었다. 그의 관심사는 디지털 특유의 상호작용, 자연, 실험적인 문자표현 등을 다루는 양방향성미디아 작품들을 창작하고 전시하는 일이다. Nimoy는 창의적 교육법, 훌륭한 통신법과 참되고 진실한 노력에서 진가를 찾는 사람이다. 그는 UCLA의 미술과 건축 대학(School of Arts and Architecture)에서 디자인과 미디아예술(Design and Media Arts)의 학사학위를 받았으며 그때 그의 전공은 디지털 문화와 기술(digital cultures and technologies)이었다. 1999년 UCLA 학부생일 때에 MIT의 심미학과 전산그룹이 주도하는 미디아 연구소에 학부생 방문연구원으로 활동한 바가 있으며 이때 Ben Fry와 Casey Reas를 만나 함께 일하기도 하였다. _

-

website

Benjamin Fry 는 MIT의 미디아 연구소의 박사과정에 적을 두고 있다. 그의 연구는 동적인 정보 소스에서 나오는 대량 데이터를 비쥬얼라이즈하는 방법에 집중되어 있다. 그의 연구는 분산 과 순응이라는 시스템 개념을 활용하여 입력되는 데이터에 응답하는 유기적 표현을 형성하는데에 주력하고 있다. 이 작업은 현재 인간 게놈의 방대한 데이터를 나타내는 게놈 지도 제작법에 응용할 목적으로 수행하고 있다. John Maeda가 주도하는 MIT의 심미학과 전산그룹의 멤버이기도 한 그는 Carnegie Mellon 대학교 디자인 대학에서 학부과정을 마쳤다. 그의 주전공은 그래픽스 디자인이고 부전공은 전산과학이었다.

website

Casey Reas 는 이태리 북부에 새로 생긴 Interaction Design Institute Ivrea 의 부교수로 재직하고 있다. (현재는 UCLA 교수 –역자 주) 그의 연구는 생물학적 계와 자연계를 다양한 디지털 미디아를 통해서 어떻게 추출 표현할 수 있는가에 집중되어 있다. 그의 디지털 미디아 개념에는 software art, digital prints, and responsive installations 등이 포함된다. John Maeda 가 주도하는 MIT의 심미학과 전산 그룹(Aesthetics and Computation Group (ACG))의 멤버이기도 한 Casey 는 2001년 이그룹에서 에서 석사학위를 받았다. , Casey는 유럽 아시아 미국등지를 다니며 강연과 전시를 해 왔다. 최근에 그의 작품은 American Museum of the Moving Image, Ars Electronica, Interaction01 in Ogaki, New York Digital Salon, Museum of Modern Art, P.S.1과 Siggraph2000 등지에서 전시되기도 하였다.

website

-

-

---

last updated October 31, 2005