Scroll, Pinch Zoom이 가능한 UIScrollView

Created	@July 26, 2022 9:04 AM
Platform	iOS
Theme	Library
Status	작성 완료
Writer	정준현
비고	

1. Description

> iOS에서 UIScrollView 내부에 들어가는 Content의 크기가 16000px을 넘어설 때 메모리를 과도하게 사용한다. 메모리 이슈를 해결하고자 SlideView 3개만을 활용하여 스크롤 될 때 SlideView의 위치를 옮기는 식으로 커다란 View를 스크롤하는 것처럼 보이게 하는 식으로 구현하였다. 또한, 핀치 줌을 통해 사용자가 보고 싶은 그래프의 영역을 설정할 수 있게 하였다.

https://github.com/weekyear/ScrollableStackView_iOS

파일 및 용어 설명

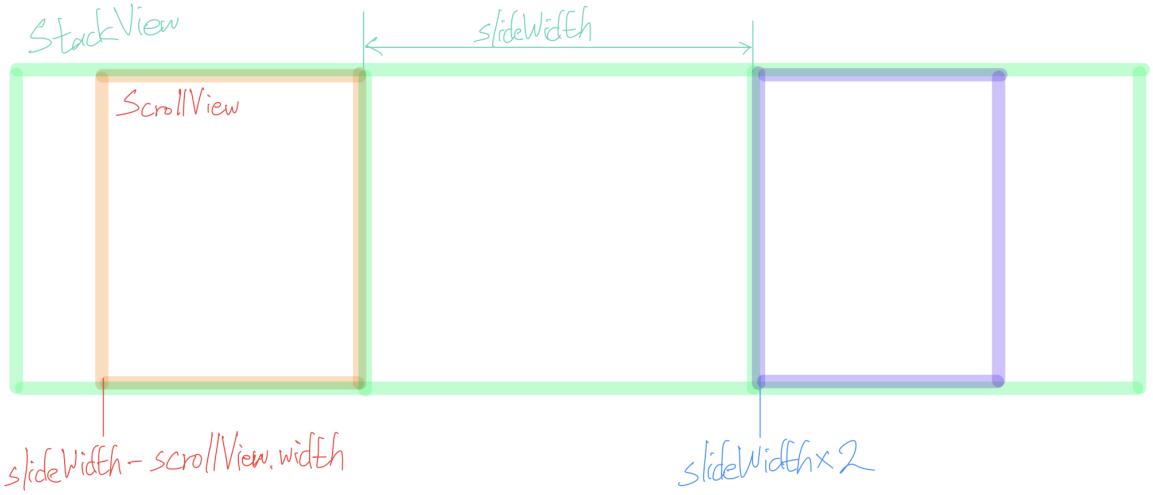
1. GraphPoint : Point나 Path를 그리기 위한 정보가 담겨져 있는 클래스, 외부에서 받아온 DataPoint를 GraphPoint로 환산해서 그림을 그린다.

2. BezierPathAlgorithm : Bezier path에 활용되는 Control Point를 계산하고 Bezier Path를 생성
3. GraphConfig : 외부에서 Graph를 그리기 위한 설정을 제어하기 위한 클래스
    - pointSize : 점의 크기
    - lineSize : 선의 크기
    - paddings : 그래프의 paddings
    - minLineGapSize : 핀치 줌 될 때 LineGapSize 최솟값
    - maxLineGapSize : 핀치 줌 될 때 LineGapSize 최댓값
    - data
4. LayoutCalculator : 그래프를 그리기 위해서 필요한 값의 계산 로직이 담긴 클래스
    - sectionMinY : 그래프 그려지는 영역의 최소 y 위치
    - sectionMaxY : 그래프 그려지는 영역의 최대 y 위치
    - sectionMinValue : 그래프 그려지는 영역의 최소 y 위치에 해당하는 값
    - sectionMaxValue : 그래프 그려지는 영역의 최대 y 위치에 해당하는 값
    - drawWidth : 그래프가 그려지는 전체 너비
    - slideWidth : 하나의 SlideView가 가지는 너비
    - totalSlideNum : DrawWidth를 커버하기 위한 가상 SlideView의 개수
    - graphPointsList : GraphPoint의 리스트의 리스트, 하나의 그래프에서 여러 개의 선을 그려야 할 수도 있기 때문에 이중 리스트 형식으로 되어 있다.
5. ScrollableStackView : UIScrollView를 상속받음, SlideView를 활용하고 핀치 줌이 가능하다. UIScrollView 내부에 StackView가 있고 StackView에 SlideView를 쌓아서 활용하고 있다.
    - targetIndex : 핀치 줌의 중심이 되는 인덱스
    - prevXOfTargetIndex : 핀치 줌이 시작 될 때 targetIndex의 화면 상 x 좌표
    - lineGapBegan : 핀치 줌이 시작될 때 lineGapSize
    - slideViewNum : 실제 SlideView를 몇 개 배치할 것인지 여부
6. SlideView : ScrollableStackView의 StackView에 들어갈 점과 선이 그려지는 View
    - slideNum : 가상의 SlideView 리스트 중 몇 번째 SlideView인지에 대한 정보

2. 동작 원리

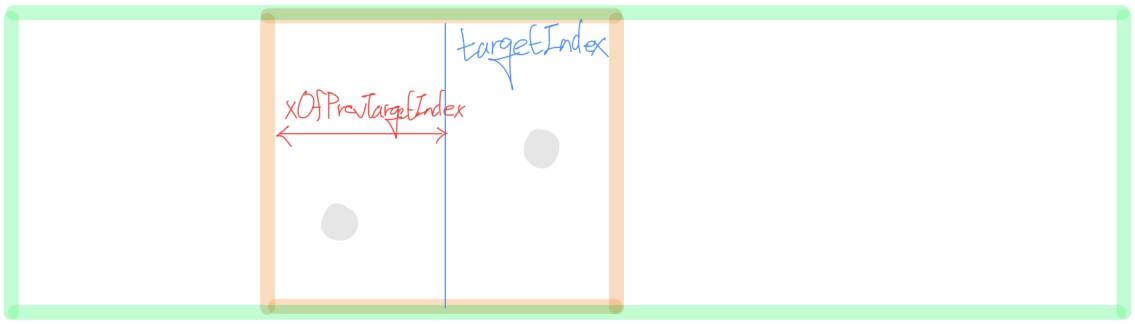
1. SlideView 스크롤 원리

1. UIScrollViewDelegate를 구현해서 scrollViewDidScroll 함수를 통해 스크롤 이벤트를 받는다.
2. 핀치 줌 중에도 scrollViewDidScroll 이벤트가 호출되는 데 핀치 줌 중에는 스크롤이 되지 않도록 해야 한다.
3. ScrollView 내부에 StackView가 있고 StackView 내부에는 3개의 SlideView가 있다. 초록색 사각형 하나의 너비가 slideWidth이다.
4. 오른쪽의 SlideView가 왼쪽으로 옮겨져야 할 때는 주황 사각형의 위치가 되었을 때 동작한다. (scrollView.x < slideWidth - scrollView.width)
5. 왼쪽의 SlideView가 오른쪽으로 옮겨져야 할 때는 보라 사각형의 위치가 되었을 때 동작한다. (scrollView.x > slideWidth * 2)



2. 핀치줌 원리

1. UIPinchGestureRecognizer를 활용하여 구현하였다. sender의 상태는 began(시작할 때), changed(움직일 때), ended(끝날 때) 3가지 상태를 활용한다.
2. 핀치 줌을 시작할 때, targetIndex, prevXOfTaregetIndex, lineGapBegan을 저장하고 시작한다. (회색 동그라미는 두 손가락 위치)

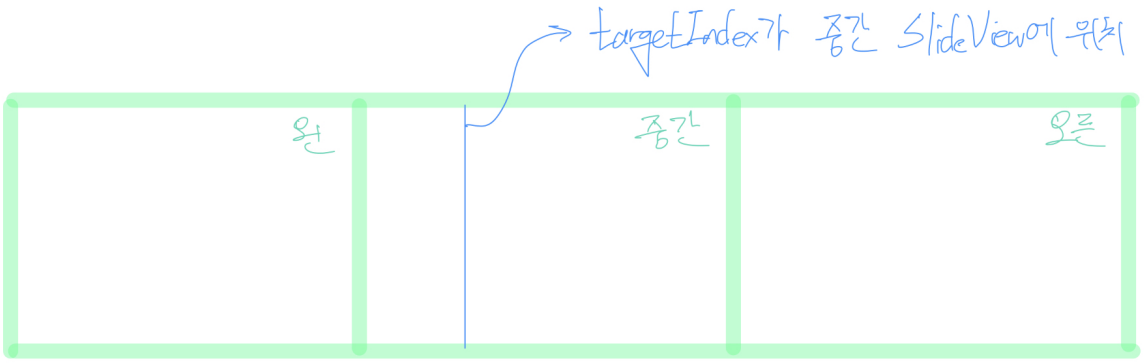


3. 핀치 줌이 진행될 때 sender의 scale을 활용해서 newLineGapSize를 계산하고 핀치 줌이 될 조건을 설정한다. (자세한 구현 사항은 코드 참고)

```
let isZoomOut = CGFloat(calculator.lineGapSize) - newLineGapSize > pinchSensitivity
let isZoomIn = newLineGapSize - CGFloat(calculator.lineGapSize) > pinchSensitivity

let isNotMaxLineGapSize = calculator.lineGapSize <= config.maxLineGapSize
let isNotMinLineGapSize = calculator.lineGapSize > config.minLineGapSize

if ((isZoomIn && isNotMaxLineGapSize) || (isZoomOut && isNotMinLineGapSize)) {
    // LayoutCalculator의 calcualte를 다시 수행
    calculator.calculateForRedraw(newLineGap: newLineGapSize)

    // SlideViews의 SlideInfo를 수정
    slideViews.forEach { curSlideView in
        calculateSlideInfo(xOfTargetIndex: CGFloat(calculator.findXByIndex(index: targetIndex)))
    }
    // 핀치 줌이 끝났을 때 contentOffset.x를 다시 잡아주기
    setOffsetXWhenPinchingZoom()
}
```

4. 핀치 줌이 진행 될 때 수행할 동작은 크게 3가지다
    a. LayoutCalculator의 calcualte를 다시 수행 : drawWidth, totalStackNum 등의 값을 다시 계산
    b. SlideViews의 SlideInfo를 수정
    c. 핀치 줌이 끝났을 때 contentOffset.x를 다시 잡아주기

> 4-b. SlideViews의 SlideInfo를 수정할 때 targetIndex를 가지고 있는 SlideView가 중간에 위치하도록 SlideView를 배치한다. 하지만, targetIndex가 0번째 SlideView나 마지막 SlideView에 위치한다면 왼쪽 SlideView나 오른쪽 SlideView에 위치시킨다.



5. 핀치 줌이 끝날 때 다음 3가지 동작을 수행하고 완료한다.

```
case .ended:
    // Pinch Zoom이 끝났을 때 SlideView의 위치를 조정하고 Offset X를 다시 잡는다.
    controlSlideViews(self)
    setOffsetXWhenPinchingZoom()
    isPinchZoom = false
    break
```

3. Documentation(사용법)

> 본 라이브러리는 그래프 기획에 영향을 받지 않는 범위 내에서 스크롤과 핀치 줌만 구현되어 있으므로 추가로 구현해야 할 사항이 많다.

1. 프로젝트를 받아온 다음에 6가지 파일을 새로운 그래프 라이브러리 프로젝트로 옮긴다.
2. 그래프의 기획에 맞게 구현 사항을 추가하거나 수정하여 사용한다.

추가 구현 사항 예시

- setOffsetXToRight : 그래프가 처음 보일 때 최근값을 보이게 구현해야할 경우 구현
- x 축에 나타나는 범위가 기간이 정해져 있을 때, GraphPoint간의 간격이 일정하지 않을 수 있다. 다음과 같이 구현하면 된다.
    1. 그래프 전체 영역에서 첫 번째 점이 찍혀야할 x 좌표(xOfFisrst)에 해당하는 날짜를 구하고 마지막 점이 찍혀야 할 x좌표(xOfFinal)의 날짜의 간격(일 수 차이)을 구한다.
    2. (xOfFinal - xOfFirst) / (일 수 차이)를 하면 lineGapSize를 구할 수 있다.

3. 각 DataPoint가 가지고 있는 날짜와 그래프에 나타나는 첫 번째 날짜의 간격을 구하면 인덱스가 나오기 때문에 xOfFirst + lineGapSize * (날짜 간격)를 하면 GraphPoint의 x좌표를 구할 수 있다.

4. Licnense

- 없음