TexTricot 4.0 인수인계

인계자 : 김태훈

인수자 : 정환석

일자 : 11. 04.

목차

내용

목차 2

머리말 4

트리코트 및 더블라셀에 대하여 5

트리코트란 5

더블라셀이란 5

직물의 분류 5

트리코트의 설계 5

가이드바 6

체인링크 6

더블라셀의 설계 7

참고 자료 7

Visual Studio 2015 로 마이그레이션 계획 8

프로젝트 세팅 : 8

프로젝트 TexTricot : 8

목적 : 8

사용 라이브러리 : 8

Manager 구성 : 8

Forms 구성 : 10

프로젝트 TexTricot 3.2 : 12

목적 : 12
프로젝트 TexTricotControls : 12
목적 : 12
Form 구성 : 12
Util 구성 : 15
프로젝트 TexTricotCLR : 15
목적 16
사용 라이브러리 : 16
TCProject 구성 : 16
TCManager 구성 : 17
프로젝트 TexTricotCore : 17
TCProject 구성 : 18
TCSimulation 구성: 18
TCProjectIO 구성 : 19
더블 라셀 계획 20
설계 20
시뮬레이션 20
KMO 파일 분석 20

머리말

10 년동안 C++Builder5.0 이라는 IDE 위에서 짜여진 코드를 유지 보수하기에는 이미 현존하는 C++Builder 개발자가 없으며, 심지어 최신 C++Builder 인 RAD Studio 마저 이 버전의 프로젝트를 지원하지 않는 다는 것을 확인한 뒤, 3D 개발 환경 및 디버깅을 위해 최신 RAD Studio 나 Visual Studio 로 이를 마이그레이션 해야한다고 결정하였다. 이에, Visual Studio 가 현재 Window 표준 개발 환경으로 자리 잡고있는 만큼 디버깅이나, 라이브러리를 추가하는데 도움이 될 만한 정보가 더 많을 것이라 생각되어 Visual Studio 로 마이그레이션을 시작하였다.

Visual Studio 의 Winform 을 사용하려면 C#이라는 언어 제약이 필요하나, 만약 시뮬레이션이라는 무거운 작업을 하게 될 경우, C++로 짜야 하는 문제점이 생긴다. 또한 현재 사내에 C#을 다룰 수 있는 사람이 많지 않다고 보아 이를 해결하기위해 C++로 프로젝트를 다루고, C#으로는 시각적인 요소만 다루게 끔 구조를 짰다. 또한, 다른 폼을 참조하지 못하게 하여, 다른 Tex3D 와 같은 업데이트 및 참조 지옥에 빠지지 않게 하였다. 짰던 코드를 짜야한다는 단순작업이 좀 늘었지만, 프로젝트가 진행되도 유지 보수가 가능하게끔 하기 위해 이런 방법을 택하였다.

생애 첫 인수인계 문서라 부족한 점이 많겠지만, 이 프로젝트가 잘 되길 바라며 글을 마치겠다.

김태훈

트리코트 및 더블라셀에 대하여

트리코트란

트리코트란 경편물의 일종으로 수영복이나 카시트 등을 생산하는데 주로 사용된다. 빠른 속도로 뽑아내는 트리코트용 기계를 사용한다.

더블라셀이란

더블라셀이란 경편물의 일종으로 카시트나 신발을 생산하는데 주로 사용된다. 트리코트보다 느린속도로 뽑아내는 라셀용 기계를 사용하지만, 가이드바가 뒷면 및 파일사를 생성하는데 들어가 라셀과는 다른 설계방식을 갖는다.

직물의 분류

섬유 원단 - 직물

- 편물 - 위편
  - 경편 - 트리코트
    - 라셀
    - 더블라셀
    - 그외

트리코트의 설계

트리코트 설계를 이해하기 위해서는 기본적으로 가이드바와 체인링크에 대해 이해해야 한다. 포토샵을 생각해볼때 가이드바란 하나의 레이어에 해당하며, 체인링크는 해당 레이어에 대응되는 그림 정보라고 생각하면 된다. 수편기를 떠올리면 된다. 자세한건 아래에 설명해 두었다.

가이드바

가이드바는 실바늘에 실을 걸어 스킬 바늘이 실을 엮어갈 수 있게끔 좌/우로 이동하는 가로로 긴 장치이다. 가로로 길게 실바늘이 무수히 늘어져 있으며, 그 실바늘마다 실을 넣고, 동시에 좌/우로 움직이게 한다. 그렇게 움직이는 도중, 스킬 바늘이 이 실을 엮어가는 방식이다.

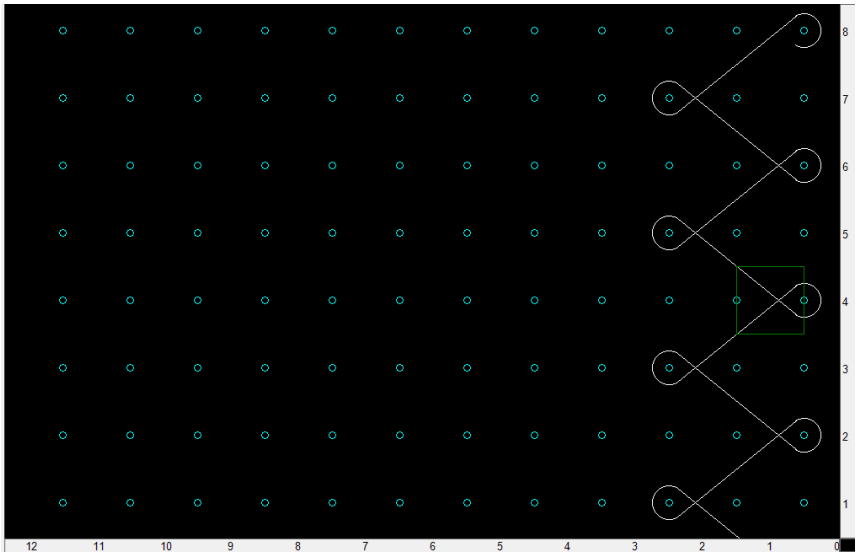
트리코트기계의 국내 트리코트의 경우 현재 7 개의 가이드바가 있는 기계가 있는 것이 확인되었으며, 이 가이드바가 많아질수록 설계가 다양하고 복잡해지게 된다. 기존 TexTricot3.2 는 가이드바 = 레이어 의 개념을 적용하였다.

트리코트 가이드바의 경우 Threading 정보를 설정할 수 있다. 이는 어느 실바늘에 실을 거는지에 대한 정보이다.

체인링크

체인링크는 트리코트 가이드바가 좌/우로 이동하는 정보이다. 따라서 트리코트 가이드바의 경우, 단 하나의 체인링크 정보를 갖는다.

체인링크는 코스라는 정보로 이루어져 있다. 코스는 X1 에서 실이 들어가고, X2 로 실이 통과하는 정보이다. 아래의 래핑정보는 (23 10) * 3 으로



아래서부터 x 축으로 2 번위치로 실이 올라가고 3 번위치로 내려온 뒤 다음 코스에서 1 번 위치로 실이 올라가고 0 번위치로 실이 내려오는 것을 볼 수 있다. 따라서 23

10 으로 쓰고 이것이 3 번 반복되었으므로 (23 10) * 3 으로 적는다. 이때 2 는 첫번째 코스의 X1 이고 3 은 첫번째 코스의 X2, 1 은 두번째 코스의 X1, 0 은 두번째 코스의 X2 로 읽는다. 3 Lapping 형태로 표현될 경우 해당 코스가 나가는 방향도 정해야 한다. 이는 다음 코스의 X1 정보를 이용해야 한다.

더블라셀의 설계

더블라셀은 트리코트에 비해 느린 라셀 기계를 사용해 만들어진다. 가이드바가 7 개인 7 바 라셀기계에서 1 번과 2 번바를 이용해 앞면을, 3 번과 4 번바를 이용해 파일사를, 7 번바를 이용해 뒷면을 짜게 시킬 수 있다. 앞면 / 파일사 / 뒷면의 순서는 바뀔 수 없으나 각 면에 사용되는 가이드바의 수는 어느정도 바뀔 수 있다. 하지만 각 기계마다 바늘이 접근 가능한 가이드바의 번호 최대치가 다르거나 파일바의 한계로 인해 설계가 불가능한 설계도면도 나올 수 있다.

트리코트와 같이 체인링크를 이용해 설계한다 (라셀 및 라셀 자카드는 이미지에 그리드를 그려 설계하는 것으로 알고있다). 이때, 뒷면 및 파일사는 단순한 체인링크가 사용되어왔으나, 이 설계 프로그램으로 더 다양한 설계를 할 수 있어야 한다.

참고 자료

이전 TexTricot 문서 : 3.0 버전과 2.3 인수인계 문서, 초기버전문서로 총 3 종류가 있다.

트리코트 개발 보고서

Visual Studio 2015 로 마이그레이션 계획

프로젝트 세팅 :

1. 프로젝트를 W:₩TexTricot4.0 에 두어 W:₩TexTricot4.0₩Data, W:₩TexTricot4.0₩TexTricot 가 있도록 한다.
2. Visual Studio 2015 를 이용해 W:₩TexTricot4.0₩TexTricot₩TexTricot.sln 을 연다.
3. Solution Explorer 에서 TexTricot 프로젝트를 우클릭하여 Properties 창을 연다.
4. Reference Path 에 W:₩TexTricot4.0₩TexTricot₩Library₩dockpanelsuite_2.10₩ 를 추가한다.
5. TexTricot3_2 에도 마찬가지로 적용한다.
6. 솔루션 전체를 리빌드한다.

프로젝트 TexTricot :

목적 :

Tricot 및 더블라셀을 설계하기 위한 Winform 프로젝트이다.

사용 라이브러리 :

DockPanelSuite, TexTricotControls, TexTricotCLR

Manager 구성 :

매니저들은 Singleton 으로 작성되어있다. 따라서 ~~~Manager.Inst 로 접근하게끔 설계하였다. 만약 프로젝트가 여러 개라던지 MainForm 을 여러 개 띄운다던지 등 다양한 기능의 추가에따라 Manager 가 여러 개가 필요하게 되면 그에 맞춰 알맞은 Manager 를 가져올 수 있게끔 인터페이스를 수정하는 것을 추천한다.

ActionManager

유저의 조작에 따라 **ProjectManager** 내의 데이터를 수정하는 관리자이다. 이때 데이터의 수정은 TexTricotCLR 내의 **IAction** 인터페이스를 상속받은 객체에 의해 이루어져야 한다.

알맞은 **IAction** 을 상속받는 액션를 생성한 뒤, **DoAction<T>()**을 통하여 해당 액션을 실행시키며, 액션이 성공적으로 실행되었으면 각 폼이 등록한 listener 에 이벤트를 날린다. 폼은 이 이벤트를 받아 자신의 자식 컨트롤들을 업데이트 할 것이다. **UnDoAction()**을 통하여 최근 실행된 액션부터 undo 할 수 있다.

AddEventListener<T>()를 통하여 알맞은 이벤트를 등록하거나, **RemoveEventListener<T>()**를 통하여 등록된 이벤트를 해지할 수 있다. 이벤트가 등록된 상태일 때, 타입 **T** 에 따라 불릴 리스너들이 결정된다.

유저의 행동 한번이 여러 개의 액션을 실행시키는 경우, **StartTransaction()**과 **EndTransaction()**을 통해 하나의 액션으로 취급할 수 있다. 이렇게 Transaction 으로 취급되는 액션들은 undo 시 한번에 undo 가 되거나 undo 가 실패할 시 액션을 다시 실행시켜 원상태로 복귀시킨다.

IAction 에 대한 자세한 정보는 TexTricotCLR 프로젝트를 참조하길 바란다.

DockFormManager

창을 띄우거나, 끌 때 사용된다. **OpenForm<T>()**를 통해 Form 을 열고, **OpenForm<T>()**를 통해 Form 을 닫으며, **ToggleForm<T>()**을 통해 On/Off 를 전환할 수 있다.

InputManager

IsKeyPressed()를 통해 현재 키보드에서 눌린 Key 를 폴링 하거나 **Is~~Input()**을 통해 사용자 입력의 포맷을 체크할 때 사용된다.

ProgressLogManager

Log()를 통하여 **MainForm** 아래에 현재 실행중인 동작을 표시한다. 만약 프로그램이 Debug 모드로 빌드 되어 있을 경우, **Log()**와 **DebugLog()**는 프로그램의 실행 경로

아래 ./log.txt 파일에 조작 정보를 쓴다. **SetLogLabels()**로 등록된 라벨이 변경되지 않을 경우 **MainForm** 하단의 라벨에 표시된다.

SetMaximumProgress(), **AddMaximumProgress()**로 진행중인 기록의 수치 최댓값을 변경하며, **AddProgress()**와 **ClearProgress()**로 진행중인 기록의 수치를 변경한다. **SetProgressBar()**로 등록된 프로그레스바가 변경되지 않을 경우 **MainForm** 하단의 프로그레스바에 표시된다.

Forms 구성 :

이 프로그램에서 사용되는 대부분의 Form 은 **TexTricotControls.Forms.MainDockContent** 를 상속받아 만들어진다. **MainDockContent** 를 상속받을 시에, **MainForm** 의 **mainDockPanel** 에 부착이 가능해지며 탈착시 윈도우의 최소크기 조정과 같은 기능이 생긴다. 자세한건 **TexTricotControls** 프로젝트를 참조하면 된다.

각 폼은 1) 리스너의 등록 및 해지, 2) 사용자 입력에 대한 행동 (winform event) 3) 폼 업데이트 4) 헬퍼 와 같이 4 개부분으로 나누었다. 따라서 아래의 설명에는 해당 폼의 전체적 기능 및 중요한 부분만 설명하도록 하겠다.

MainForm

프로그램을 켤 때 가장 기본적으로 떠야 할 폼이다. 이 폼에서는 다른 창을 띄우거나 Environment 값 및 GuideBar 의 개수나 타입에 대해 조정할 수 있다.

YarnListForm

실의 대부분의 정보를 조정할 수 있다.

DataGridView 를 이용한다. 이는 **VirtualMode** 가 꺼져있을때와 켜져있을 때 행동이 다르므로 주의해서 다뤄야할 윈폼 컨트롤이다. **YarnListForm** 은 이를 항상 꺼놓고 사용한다. **VirtualMode** 란, 행의 개수가 많아져 컨트롤의 성능이 저하가 되는 것을 막기위해, 행의 수를 현재 보이는 수만큼만 생성하고, 마우스가 스크롤 되어도, 기존에 쓰던 행을 계속 쓰는 방식이다.

VirtualMode 가 꺼져있으면, **DataGridView** 의 업데이트를 사용자가 직접 Row 를 넣고, 해당 정보를 바꿔야한다. 사용자의 입력에 의해 행의 값이 변하는 경우 **CellBeginEdit** 과 **CellEndEdit** 으로 이벤트가 간다. 사용자가 행을 추가할때는 **UserAddedRow** 로 이벤트가 가며 사용자가 행을 삭제할때는 **UserDeletingRow** 로 간 뒤에 **UserDeletedRow** 로 간다. 이때, **UserDeleteingRow** 에서 지울 수 있을지 정하며, 실제로 지워지는 것은 **UserDeletingRow** 와 **UserDeletedRow** 사이이므로, 행이 지워진 뒤에 **DataGridView** 를 다시 업데이트 해주어야한다.

VirtualMode 가 켜져있으면, Update 시에 값을 쓸 수 없다. **CellValueNeeded** 이벤트를통해 해당 Cell 에 값을 써야한다. 또한 사용자 입력에 이해 행의 값이 변할 때, **BeginEdit** 과 **EndEdit** 사이에 **CellValuePushed** 이벤트가 호출되며, 이때 값이 바뀌지 않으면 **EndEdit** 에 잘못된 값이 들어간다.

ChainlinkCourseForm

각 코스의 래핑을 제외한 정보를 조작할 수 있다. 코스의 수가 최대 5000 까지 가므로 위에서 설명한 DataGridView 를 VirtualMode 로 사용한다.

ChainlinkDesignForm

각 가이드바의 코스 래핑 정보만을 설계하는 폼이다. ChainlinkDesignPanel 을 이용하며 가이드바

ChainlinkGuideForm

각 가이드바의 쓰레딩 정보를 설정(실을 넣을 바늘을 선택)할 수 있다.

ChainlinkTextForm

ChainlinkDesignForm 이 래핑을 한 개씩 설계한다면, 이 폼은 래핑을 특정 규칙으로 여러 개 설계할 수 있다. 3 가지 모드가 있는데 각각 파싱이나 표현 방법이 다르다.

TextMode 1: 선택된 가이드바의 래핑을 한 줄마다 하나씩 입력한다.

TextMode 2: 모든 가이드바의 래핑을 한줄로 표현한다.

TextMode 3: 선택된 가이드바의 래핑을 EL 형식으로 표시한다.

이때 EL 형식은 Chainlink 의 ELString 이 있는지의 여부에 따라 사용할 것인가 사용하지 않을 것 인가가 결정된다. ELString 은 다른 코스를 세팅을 하면 자동으로 초기화된다.

프로젝트 TexTricot 3.2 :

목적 :

이전 TexTricot3.2 의 UI 를 최대한 유지하면서 마이그레이션을 하기 위한 프로젝트이다. 기본적으로 TexTricot 와 같은 라이브러리를 사용하는 UI 만 다른 프로젝트라 생각하면 편하다.

프로젝트 TexTricotControls :

목적 :

TexTricotCLR 프로젝트를 사용하는데 있어서 공통적으로 사용할 것 같은 폼 및 컨트롤을 모아놓은 라이브러리이다.

Form 구성 :

ChainlinkDesignPanel

체인링크의 래핑정보를 설계하는 창이다. 상당히 많은 property 와 그에 따른 구현이 있다.

주요 property 목록 :

Zoom : 얼마나 가까이서 캔버스를 볼 지 정한다. **Zoom** 값이 높으면 크게 보이고 (돋보기) 낮으면 작게보인다.

ZoomMin / ZoomMax : Zoom 의 최소 / 최댓값을 정한다.

GridInterval : 각 그리드 격자간 픽셀 간격

CanvasMousePoint : 현재 ChainlinkDesignPanel 의 마우스 위치

GridCellMousePoint : 현재 마우스의 위치에서 가장 가까운 격자 점

GridEdgeMousePoint : 현재 마우스의 위치에서 가장 가까운 격자 모서리

GridCellAreaBegin / GridCellAreaEnd : 현재 마우스가 CellArea 모드인경우 영역으로 잡힌 격자점의 시작 ~ 끝. 어느 방향이 시작이라는 보장이 없으므로, 둘의 minX, minY, maxX, maxY 를 이용해 사각형 정보를 재구축해야한다.

AlwaysRenderMouseRect : RefreshMouseRect()를 하지 않아도 **Redraw()**시에 마우스의 사각형을 다시그린다.

CanvasBackgroundColor / CanvasGridColor / DefaultYArnColor : 색상을 나타낸다.

ViewFocusGridMin / ViewFocusGridMax : 현재 캔버스가 그리는 영역을 격자점을 기준으로 나타낸다.

ViewRangeGridMax : 캔버스의 최대크기를 나타낸다. X 는 **MaxWaleNumber** (const 값으로 256)이며 Y 는 현재 프로젝트의 Course 값이다.

렌더링 :

RefreshChainlink()와 **RefreshMouseRect()**에 의해 다시 그릴 필요가 있다는 dirty flag 를 세팅하고, **Redraw()**를 통해 다시 그린다. Chainlink 는 우선 각 격자점을 그린다. 그 후에, 각 체인링크가 반복해서 보이지 않으면 **RenderChainlink()**를 통해 그리고 반복해서 보이면 **RenderGuideBarRepeat()**를 통해 그린다.

RenderChainlink()는 한번만 그리기 때문에 성능의 저하가 거의 없으나, **RenderGuideBarRepeat()**는 여러 번 그려야 하므로 성능의 저하를 고려하여 WinGDI+가 아닌 WinGDI 를 이용한다.

RenderGuideBarRepeat()는 우선 쓰레드의 In, Out 을 조사해 몇 개의 체인링크를 그릴지 정한다. 그 후 현재 체인링크에 대한 정보를 **CreateChainlinkInfo()**를 통해

만든다. 마지막으로 체인링크에 대한 정보를 조사된 In Out 만큼의 오프셋을 주고 반복해서 그린다.

마우스 :

마우스는 **MouseRectState** 의 상태를 갖는다. Cell 은 하나의 격자 점을 중앙으로 사각형을 그리며, Edge 는 격자의 모서리를 중앙으로 사각형을 그린다. **CellArea** 는 **CellArea** 모드로 전환된 지점부터 현재 마우스까지의 영역을 사각형으로 그리며, **CellAreaSelected** 는 **CellArea** 로 전환된 지점부터 **CellAreaSelected** 로 전환된 지점까지의 영역을 사각형으로 그린다.

마우스 상태를 변경하기위해서는 **MouseState** property 를 이용하면 된다. 이 값이 변경되면 자동으로 마우스의 위치를 잡아 **GridCellAreaBegin / GridCellAreaEnd** property 를 사용할 수 있게 된다.

ConfirmDialog

만들다 말았다. 특정 기능에 대한 경고를 사용자에게 띄울 때 이 Dialog 를 이용해서 띄울 생각이었다.

DoubleBufferPanel

더블 버퍼링을 이용하여 그릴 때 깜빡임을 없애는 패널이다.

MainDockContent

MainDockPanel 에 도킹이 가능한 창들이다. 이 창들은 각 프로젝트 내의 **DockFormManager** 에서 관리되어 창이 띄워질 것이다. **DefaultMainDockState** property 를 통해 기본적으로 띄워질 위치를 지정할 수 있으나, 이는 **DockStates** property 에 적용 된 값이어야 한다.

Ruler

자를 표시한다. **VRuler** (Vertical)와 **HRuler** (Horizontal)가 있으며, 특정 범위를 지정하면 자신의 패널을 해당 범위로 채운다.

RangeMin, **RangeMax** property 와 **SetRange()**로 Ruler 의 크기를 지정하며, **FontSize** 를 통해 폰트의 크기를 정한다. 각 property 는 getter 도 구현되어 있다. 방향에 대한 값은 구현되지 않았다. 즉, **VRuler** 는 항상 아래에서 위쪽방향으로 증가하며, **HRuler** 는 오른쪽에서 왼쪽방향으로 증가한다. 구현이 필요할 시 **bool InvertDirection** property 를 생성하고 각각의 **OnPaint()**를 고치면 된다.

SimpleTextBoxDialog

텍스트박스만 있는 폼이다. Dialog 로 쓰이며, **ChainlinkDesignForm** 의 우클릭시에 뜨는 텍스트박스로 활용되었다.

Util 구성 :

WinGDI :

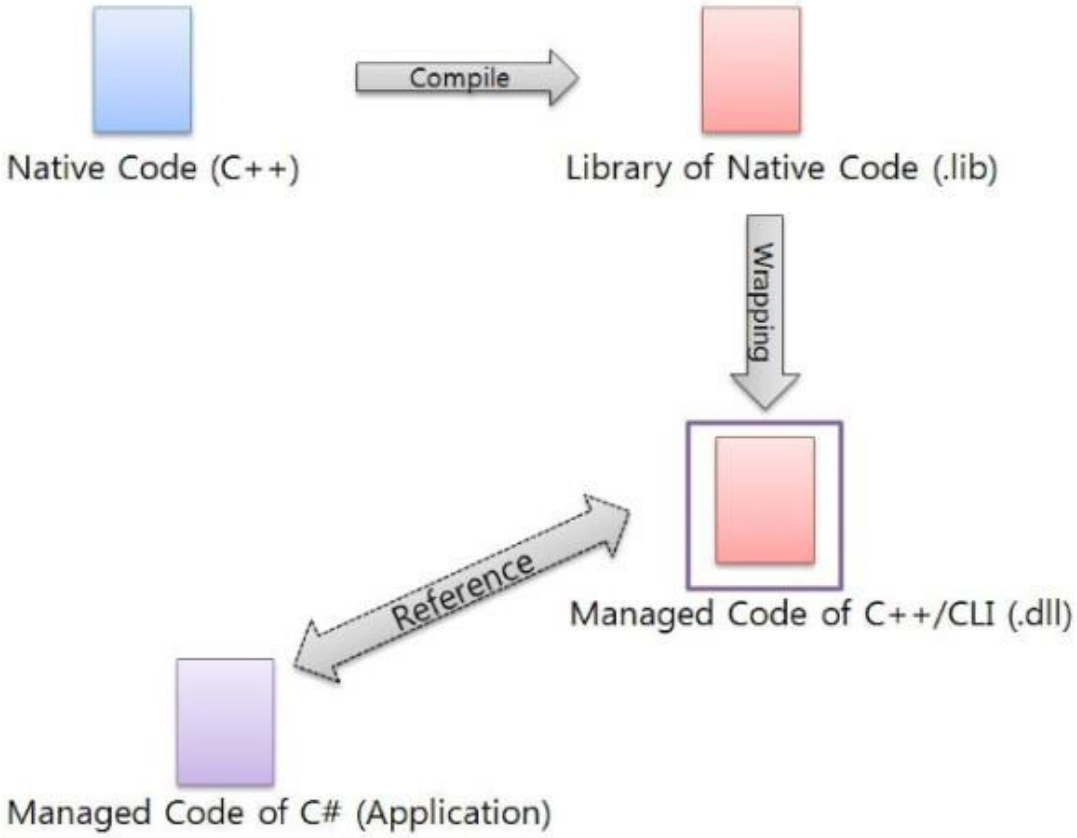
C#의 WinGDI+는 WinGDI 보다 편하게 쓸 수 있는 대신 느리다. 따라서 구 WinGDI 를 가져와서 쓰도록 래핑해준 클래스이다. WinGDI 는 펜을 특정 지점으로 이동시키고 (**MoveToEx()**), 그 지점부터 선을 그린다 (**LineTo()**)는 단순한 API 를 이용한다. 필요한 함수만 래핑하였다.

자세한 API 내용은 https://msdn.microsoft.com/en-us/library/vs/alm/dd145203(v=vs.85).aspx 를

추가적인 API 는 http://www.pinvoke.net/default.aspx 의 좌측 메뉴를 이용하여 구현하길 바란다. (ex. http://www.pinvoke.net/default.aspx/gdi32.lineto)

프로젝트 TexTricotCLR :

C++/CLR 이라는 MS 자체 언어를 사용하나, C++과 C#을 섞어 놓은 것이라 생각하면 습득하는데 긴 시간이 걸리지 않을 것이라 생각된다. 전체적으로 다음과 같은 구조로 되어있다 생각하면 된다.



목적

TexTricotCore 를 C#에서 사용하기 위해 래핑하며, **`IAction`** 을 구현하여 사용자의 입력에 대해 Undo 기능을 원활히 지원하기위한 모듈이다.

사용 라이브러리 :

TexTricotCore

TCProject 구성 :

객체의 이름이 TCCore 와 같아 컴파일이 안되기 때문에 기본적으로 API prefix 로 "TC" (Tricot)를 붙였다. 기본적인 프로젝트의 구조는 TexTricotCore 와 같은 구조를 따라간다. 새로 추가된 **`Context`** 란 클래스에 대한 것만 따로 설명하겠다.

Context

현재 프로젝트의 상황으로, 지금은 선택된 바의 정보만 들고 있으나 추후 폼간의 정보 교환이나 프로젝트와 무관한 임시 정보를 저장하는데 사용될 객체이다. 선택된 바의

정보는 **Selected~~** property 나 **GetSelectedGuideBar(), GetSelectedGuideBarAs<T>()**로 가져올 수 있다.

TCManager 구성 :

ProjectManager

프로젝트를 관리하는 클래스로, 현재 프로그램 및 프로젝트에 대한 모든 정보를 담고 있다. 다른 매니저와 마찬가지로 Singleton 으로 구현되어 있어서 정적으로 참조가 가능하다. 이 프로젝트 매니저는 **IAction** 의 파라미터로 쓰여 액션이 바꿀 프로젝트의 대상을 지칭하는데도 쓰인다.

IAction

CanDoAction() / DoAction() / CanUndoAction() / UndoAction() 을 갖는 인터페이스이다. **DoAction()** 및 **UndoAction()** 해당 기능을 실행하는 코드만 있어야 하고 **CanDoAction()** 과 **CanUndoAction()**이 실행 가능성을 조사할것이다. 이때 **UndoAction()**은 Undo 기능을 완벽히 할 수 있을 때 만 실행되어야 하며, 하나의 Transaction 에서 **UndoAction()** 이 실패하여 다시 원상복귀시킬때는 **CanDoAction()**없이 **DoAction()**이 호출될 것이다.

액션은 프로젝트의 자료구조에 맞춰 각각 구현되어 있으며, 각 액션마다 **Eventtype** property 를 이용해 세부적인 행동을 나누었다. 각 폼에서는 이 **Eventtype** 만 체크를 해도 충분히 적은 컨트롤의 업데이트를 할 수 있을 것이다. 각 액션의 세부 내용은 코드를 읽는 것을 추천한다.

개선 사항 : 인터페이스를 DoAction 하나로 합치는 것이 더 나아보인다. **bool DoAction(ProjectManager^ manager, int Mode);** 와 같이 쓰면, **DoAction()**에 대한 체크와 행동이 따로 따로 있지 않아 시간이 줄어들며, 코드도 **Eventtype** 으로 나눈 뒤 **Mode** 에 따라 나뉘므로 프로그래밍의 스코프가 줄어들 것이다.

프로젝트 TexTricotCore :

실제 프로젝트의 데이터를 저장하며, 이를 이용해 데이터를 변형하거나 가공하는 기능을 제공하기 위해 만들어진 프로젝트.

TCProject 는 프로젝트의 데이터를 구조에 맞게 저장하는 기능이 있다. TClr 을 이용해 안전하게 데이터가 바꾸어 질 것이라는 가정하에 이 자료구조는 데이터 구조의 안전성에 대한 검사를 최소로 한다. (ex. Threading 의 YarnIndex 가 3 으로 적혀있으나 실제 Yarn 은 2 개만 있을 수 있음).

TCProjectIO 는 TCProject 의 일부를 추출하거나, 특정 파일로부터 TCProject 또는 그 일부의 정보를 만들기 위해

TCProject 구성 :

프로젝트의 데이터를 저장하는 부분이다.

Project – GuideBar -> Bar 를 갖고 있다. 캐스팅을 통해 실제 Bar 의 타입으로 변형해서 써야한다.

- Chainlinks -> 체인링크의 정보를 갖고 있다.
- Yarns -> 실의 정보를 갖고 있다.
- Production -> 생산 작지 정보를 갖고 있다.

Data₩projectStructure.xml 에 맞춰서 데이터를 선언하였으므로 참고하도록 한다.

TCSimulation 구성:

TexTricot3.2 의 가장 중요한 기능이다. TexTricot3.2 의 class tricot 가 시뮬레이션을, Yarns 가 렌더링을 담당했으나, 이를 TexTricot4.0 에서 class TricotSimulation, class TricotRenderer 로 나눌 계획이다.

아직 만들지 않았지만 TricotRenderer 는 TricotSimulation 의 결과를 2D 로 렌더링 하는 객체이며, 이를 비트맵으로 뽑아내거나 주어진 비트맵에 이를 그리는 기능을 구현해야 한다. TricotSimulation 은 마이그레이션하다가 말았다.

TCProjectIO 구성 :

프로젝트나 프로젝트 내부의 일부 자료를 다른 포맷으로 바꾸거나, 다른 포맷에서 프로젝트나 프로젝트 내부의 일부 자료형으로 바꾸는 기능을 구현한다. Import / Export / Save / Load 기능을 전부 TCProjectIO 에 구현하여 후에 유지보수를 편하게 하길 바란다. 만약 일부 자료구조의 **`private`** 이나 **`protected`** 멤버에 접근해야할 경우, c++의 **`friend`** 키워드에 대해서 조사해보길 바란다.

현재는 ProjectStringIO.cpp 에, 각 체인링크를 **`wstring`** 으로 바꾸는 기능만 구현되어있다.

더블 라셀 계획

설계

더블 라셀의 설계는 기본적으로 트리코트와 같다. 아마 UI 를 바꾸고 체인링크 래핑의 즐겨찾기 기능을 만드는 일이 설계의 끝일 것이라 크게 어려운 일이라 생각되지 않는다.

시뮬레이션

2D 시뮬레이션또한 현업에서는 TexTricot3.2 를 이용하여 앞면 / 뒷면 따로 시뮬레이션을 돌린다고 한다. 시뮬레이션 되지 않는 면은 투명도를 90% 까지 올려서 매우 흐릿하게 시뮬레이션 하는 방향으로 지원하고, 앞/뒤에서 따로 따로 시뮬레이션을 해주면 되겠다. 그러나 파일이 만들어내는 당김효과는 무시할 정도가 아닌 것 같다. 이에 대한 고려도 추후에 필요할 것이다.

3D 는 OpenGL 로 하거나 DX 로 하거나 어느쪽이든 3D 를 셰이더 단위로 내려가서 그리지 않으면 성능이 안될 것 같다.

KMO 파일 분석

이 분야는 사실 분석만 정확히 되면 빠른 시간 내에 될 것이나, 분석까지 얼마나 걸릴지 알 수가 없다는 것이 문제이다. 현재 가장 좋은 방법은 EAT 사의 CAD 프로그램을 이용한 역공학으로, 프로그램을 구할 수 있으면 이를 디스어셈블을 해보거나, 그렇지 않으면 다량의 같거나 비슷한 체인링크에 대하여 kmo 파일을 추출하여 차이점이나 공통점을 이용해 어떻게 파일이 만들어 졌는지 분석하는 것이다.

현재는 .kmo 는 .zip 파일의 포맷으로, 이를 확장자만 바꾸어 압축을 풀면 keys.xml 과 암호화된 xml 파일들이 나온다는 것 까지만 분석되었다.