(12) UK Patent (19) GB (11) 2548323 (13) B

(45) Date of B Publication 17.11.2021

(54) Title of the Invention: A user interface mechanism

(51) INT CL: ***G06F 30/13*** (2020.01) ***G06F 30/12*** (2020.01)

(21) Application No: **1602256.8**

(22) Date of Filing: **08.02.2016**

(43) Date of A Publication **20.09.2017**

(56) Documents Cited:
US 20020103557 A1

(58) Field of Search:
As for published application 2548323 A viz:
Other: **No search performed: Section 17(5)(b)**
updated as appropriate

Additional Fields
INT CL **G06F, G06K, G06T**
Other: **WPI, EPODOC**

(72) Inventor(s):
Robert Reuven Franks
Christopher Alan Brunsdon

(73) Proprietor(s):
TommyTrinder.com Limited
3 Portland Place, Pritchard Street, Bristol, BS2 8RH,
United Kingdom

(74) Agent and/or Address for Service:
Rational IP Limited
81 Rivington Street, London, EC2A 3AY,
United Kingdom

01 03 17

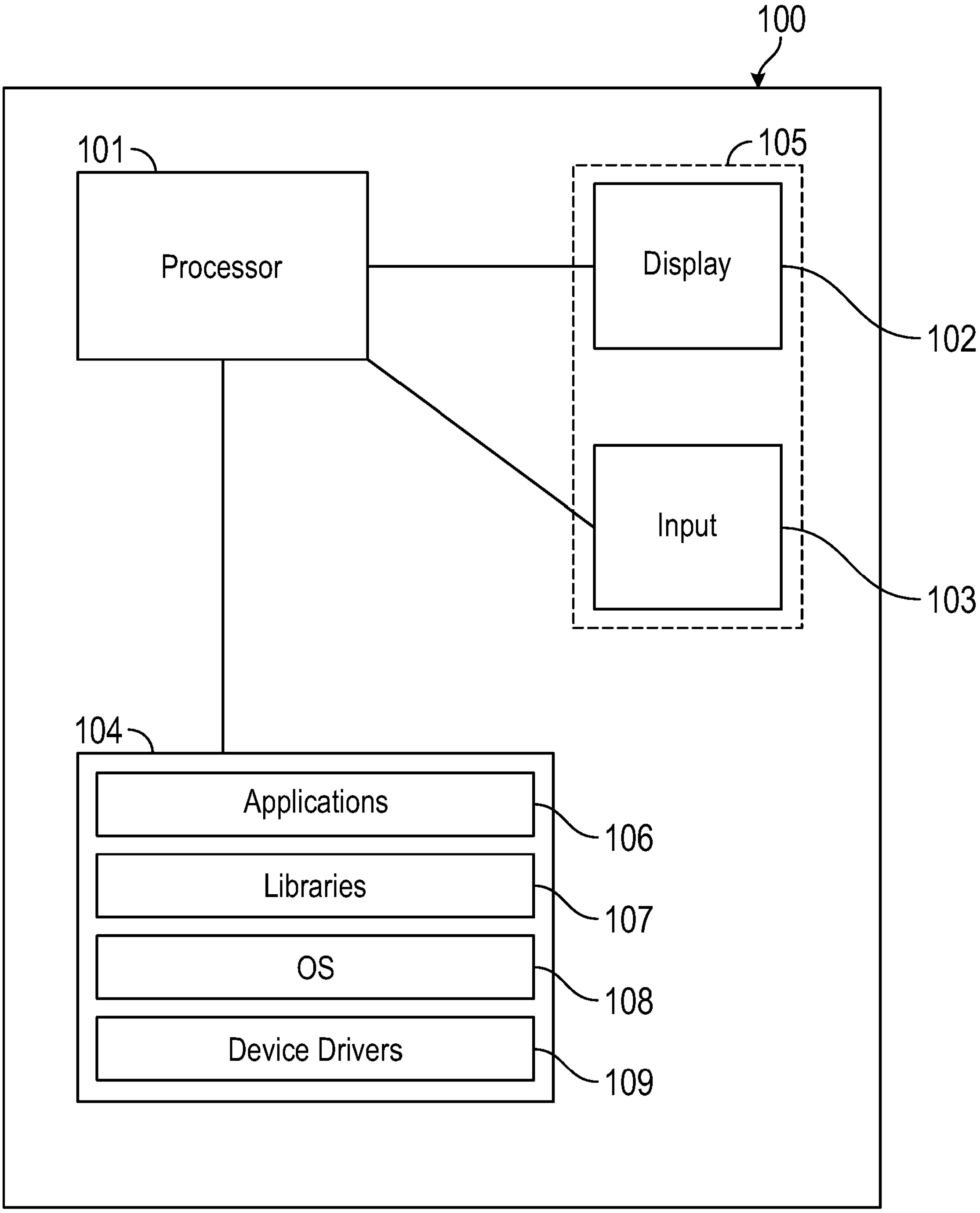
100
101
Processor
105
Display
102
Input
103
104
Applications
106
Libraries
107
OS
108
Device Drivers
109

FIG. 1

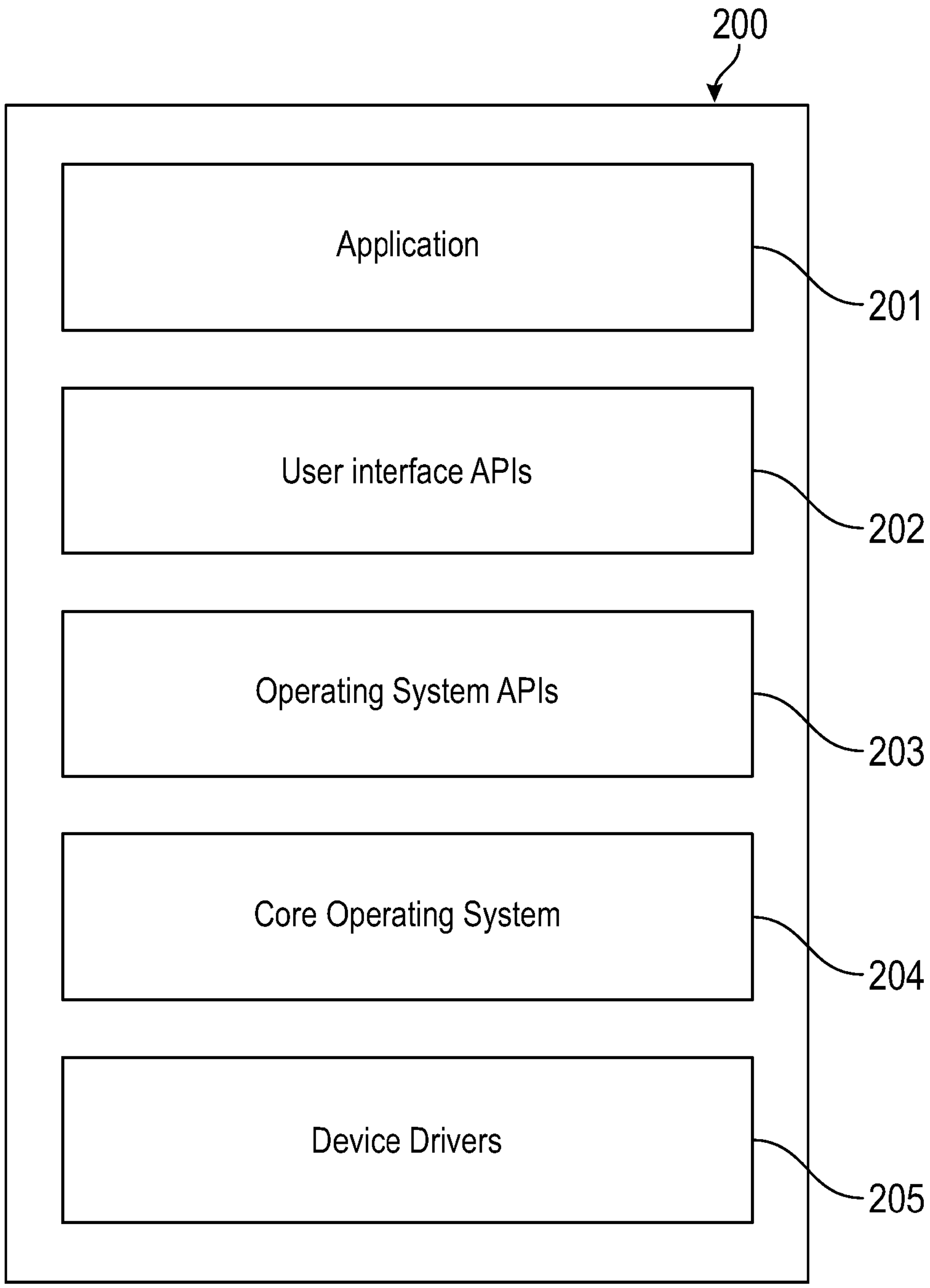
200
Application
201
User interface APIs
202
Operating System APIs
203
Core Operating System
204
Device Drivers
205

FIG. 2

01 03 17

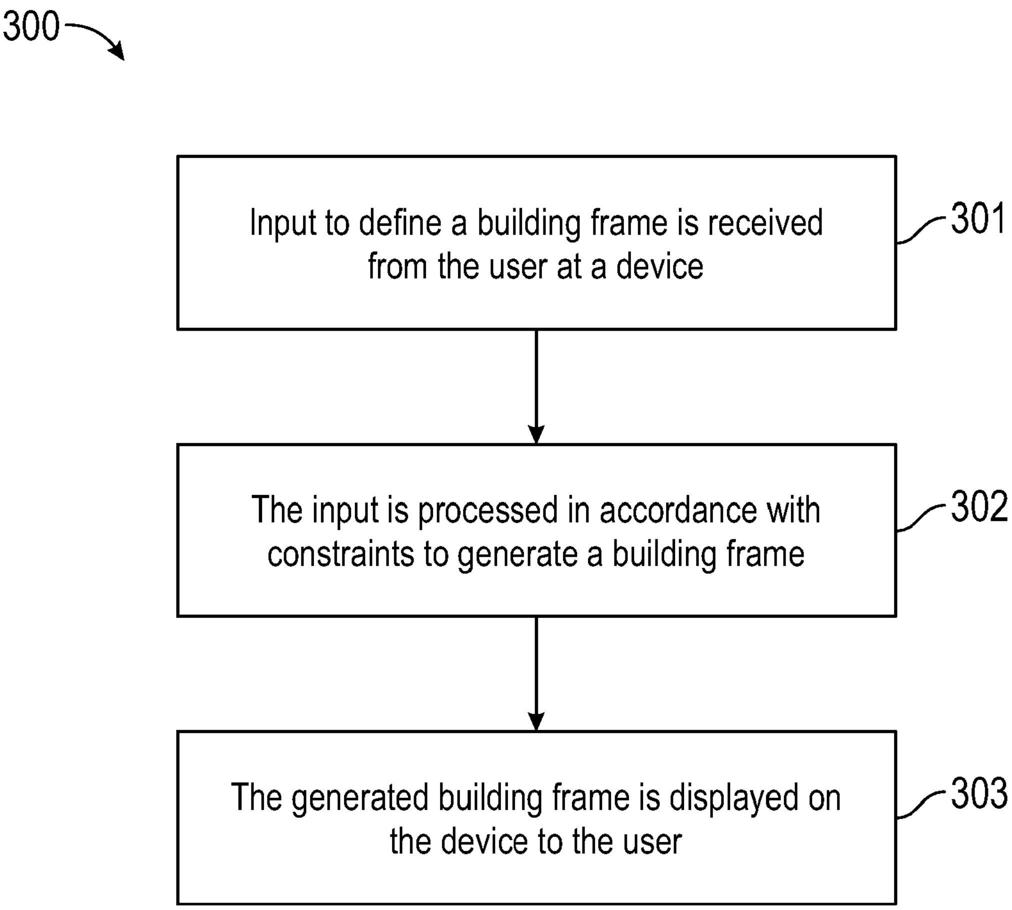
300
Input to define a building frame is received from the user at a device
301
The input is processed in accordance with constraints to generate a building frame
302
The generated building frame is displayed on the device to the user
303

FIG. 3

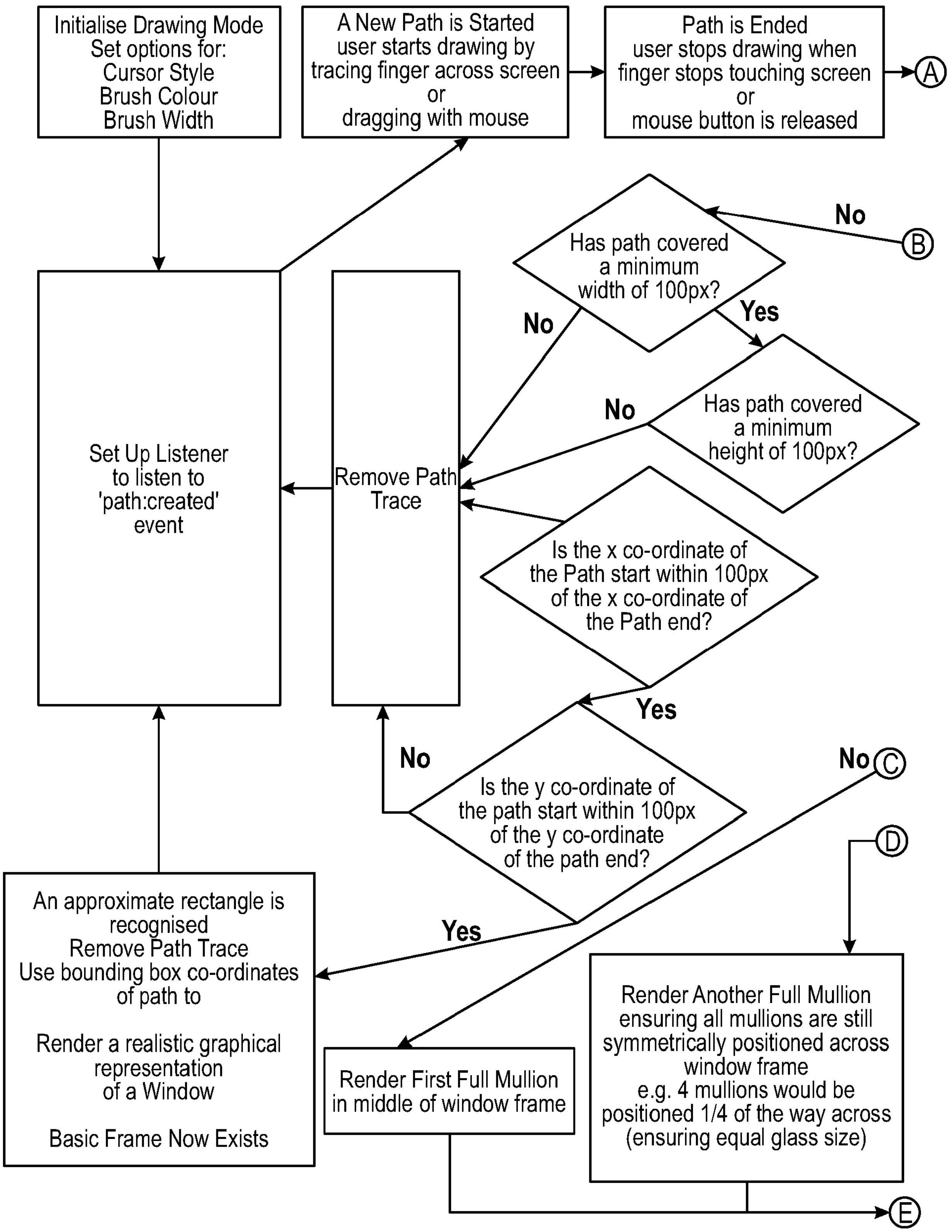
Initialise Drawing Mode
Set options for:
Cursor Style
Brush Colour
Brush Width
A New Path is Started
user starts drawing by
tracing finger across screen
or
dragging with mouse
Path is Ended
user stops drawing when
finger stops touching screen
or
mouse button is released
A
No
B
Has path covered
a minimum
width of 100px?
No
Yes
Has path covered
a minimum
height of 100px?
No
Set Up Listener
to listen to
'path:created'
event
Remove Path
Trace
Is the x co-ordinate of
the Path start within 100px
of the x co-ordinate of
the Path end?
Yes
No
Is the y co-ordinate of
the path start within 100px
of the y co-ordinate
of the path end?
No
C
D
Yes
An approximate rectangle is
recognised
Remove Path Trace
Use bounding box co-ordinates
of path to
Render a realistic graphical
representation
of a Window
Basic Frame Now Exists
Render First Full Mullion
in middle of window frame
Render Another Full Mullion
ensuring all mullions are still
symmetrically positioned across
window frame
e.g. 4 mullions would be
positioned 1/4 of the way across
(ensuring equal glass size)
E

FIG. 4

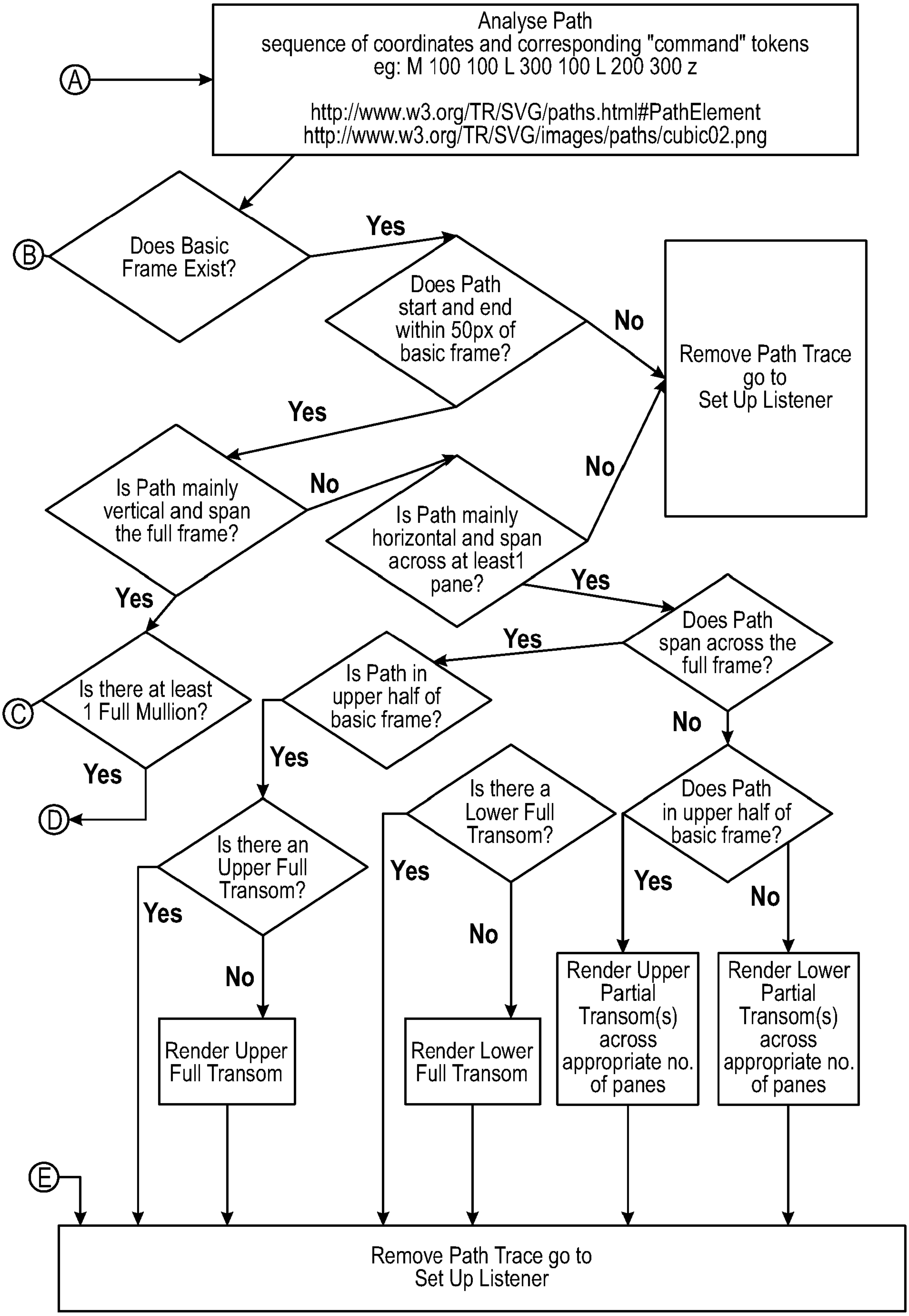
A
Analyse Path
sequence of coordinates and corresponding "command" tokens
eg: M 100 100 L 300 100 L 200 300 z
http://www.w3.org/TR/SVG/paths.html#PathElement
http://www.w3.org/TR/SVG/images/paths/cubic02.png
B
Does Basic Frame Exist?
Yes
Does Path start and end within 50px of basic frame?
No
Remove Path Trace go to Set Up Listener
Yes
Is Path mainly vertical and span the full frame?
No
Is Path mainly horizontal and span across at least1 pane?
No
Yes
Yes
Does Path span across the full frame?
C
Is there at least 1 Full Mullion?
Yes
D
Yes
Is Path in upper half of basic frame?
Yes
No
Is there an Upper Full Transom?
Yes
No
Render Upper Full Transom
Is there a Lower Full Transom?
Yes
No
Render Lower Full Transom
Does Path in upper half of basic frame?
Yes
No
Render Upper Partial Transom(s) across appropriate no. of panes
Render Lower Partial Transom(s) across appropriate no. of panes
E
Remove Path Trace go to Set Up Listener

FIG. 4
(Continued)

01 03 17

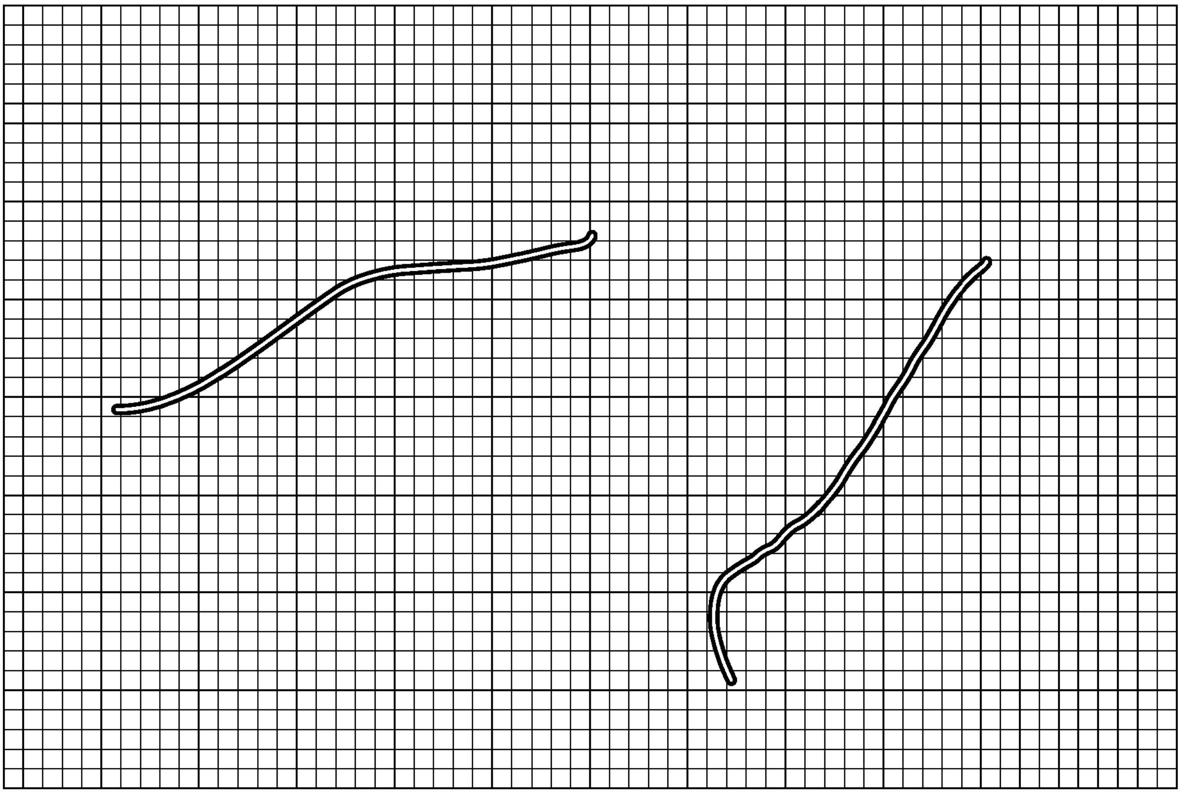

FIG. 5

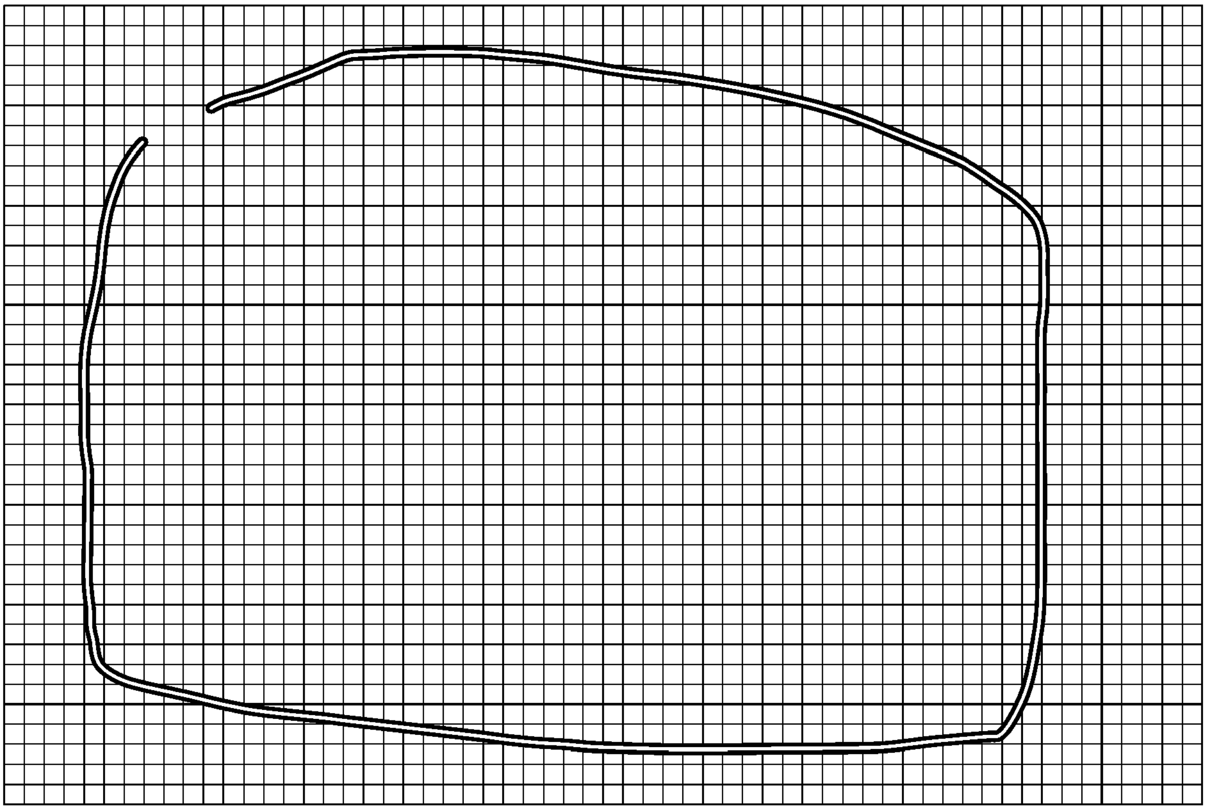

FIG. 6A

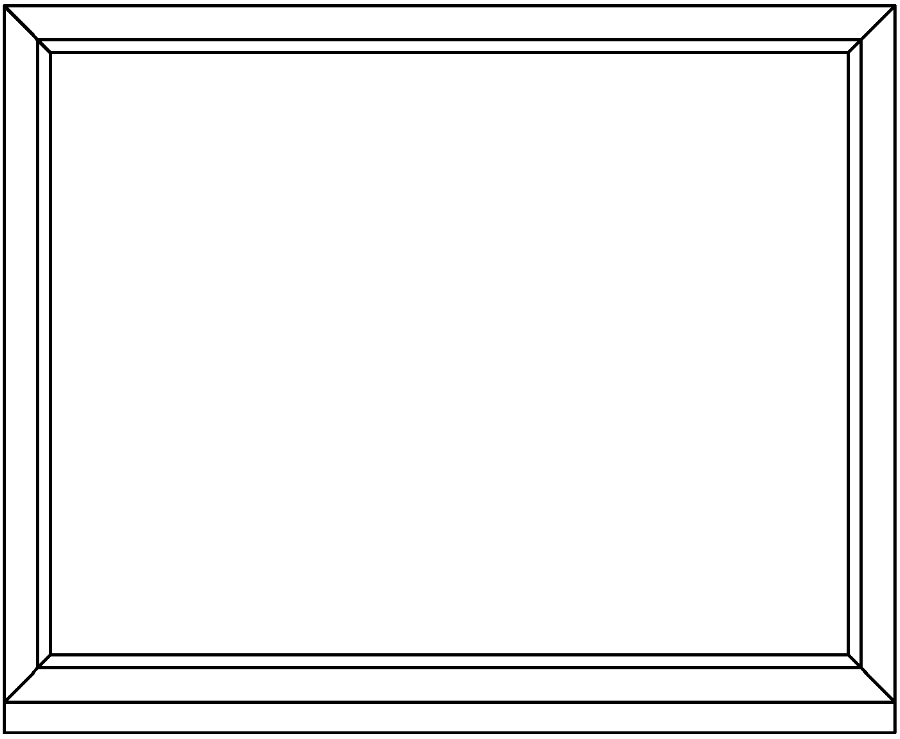

FIG. 6B

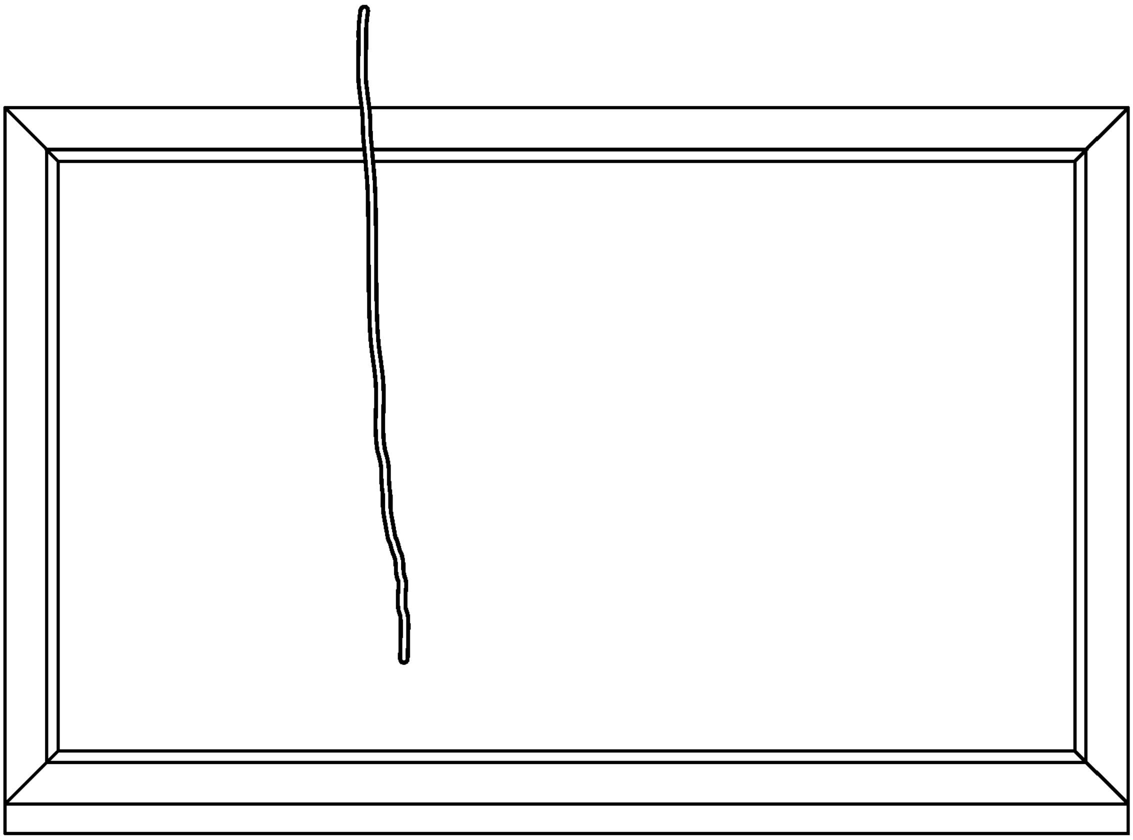

FIG. 7A

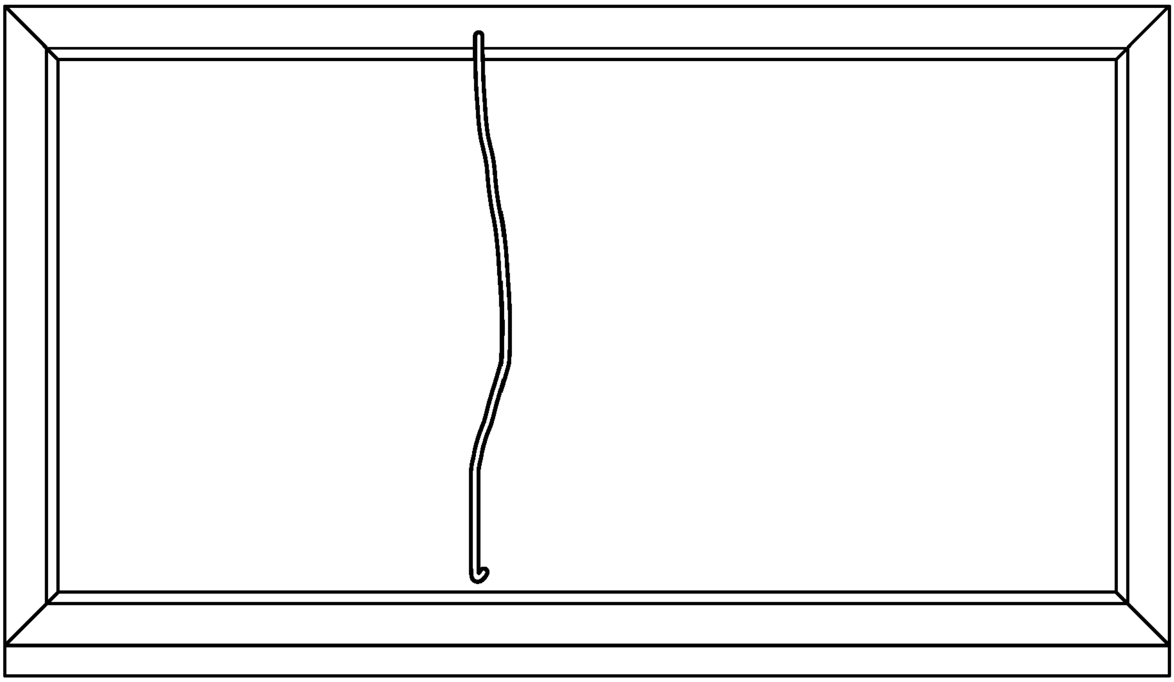

FIG. 7B

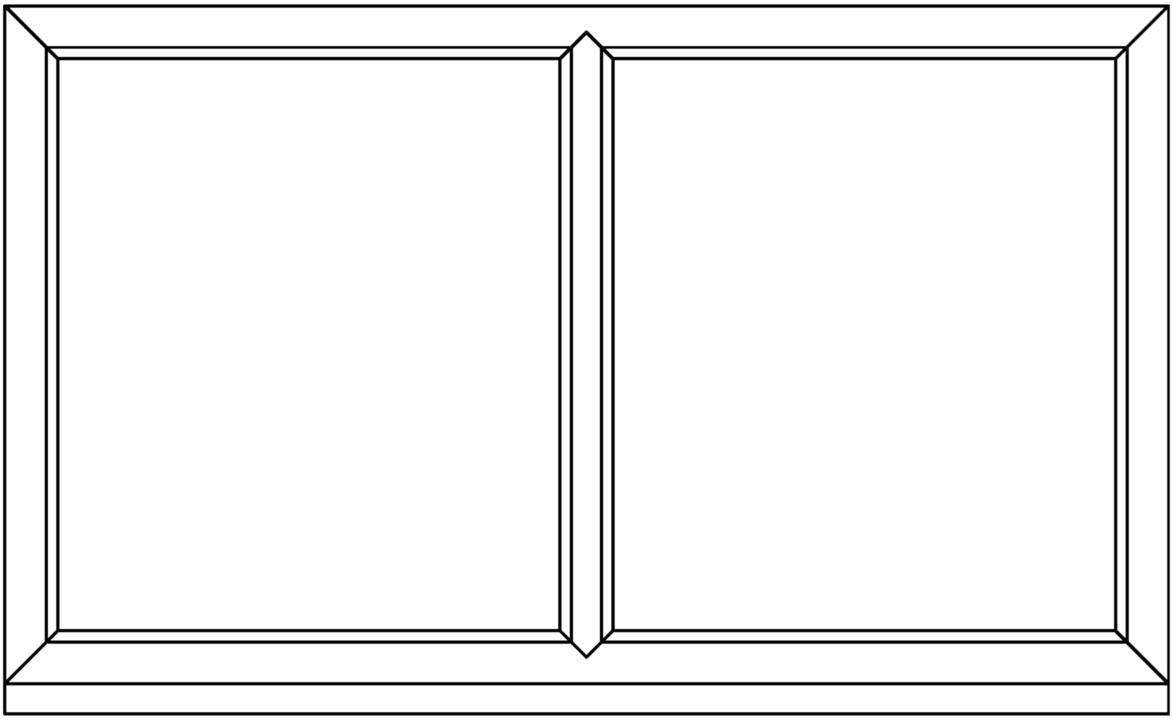

FIG. 7C

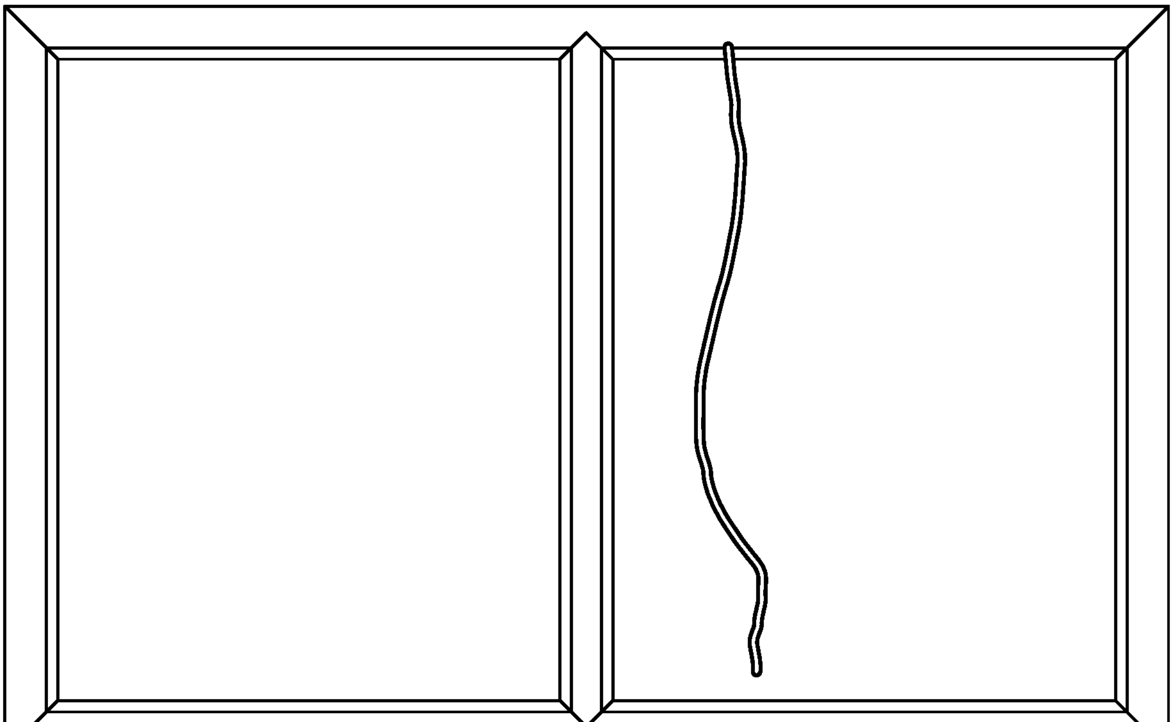

FIG. 7D

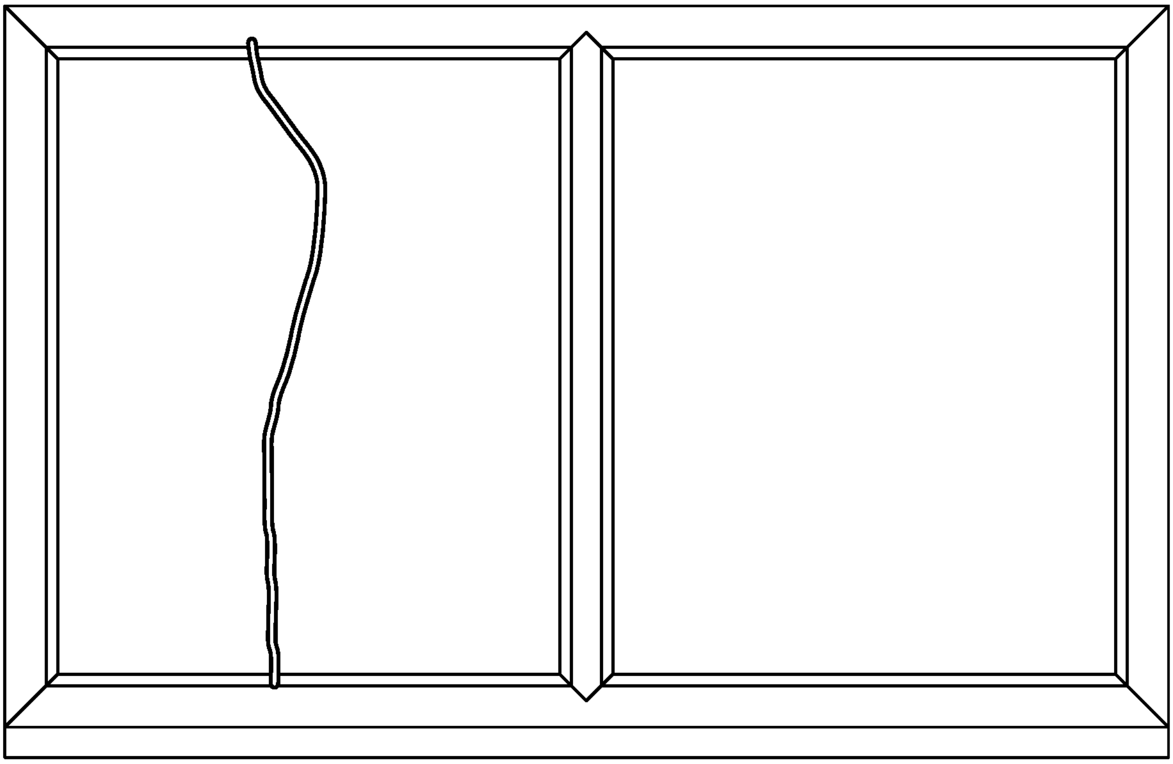

FIG. 7E

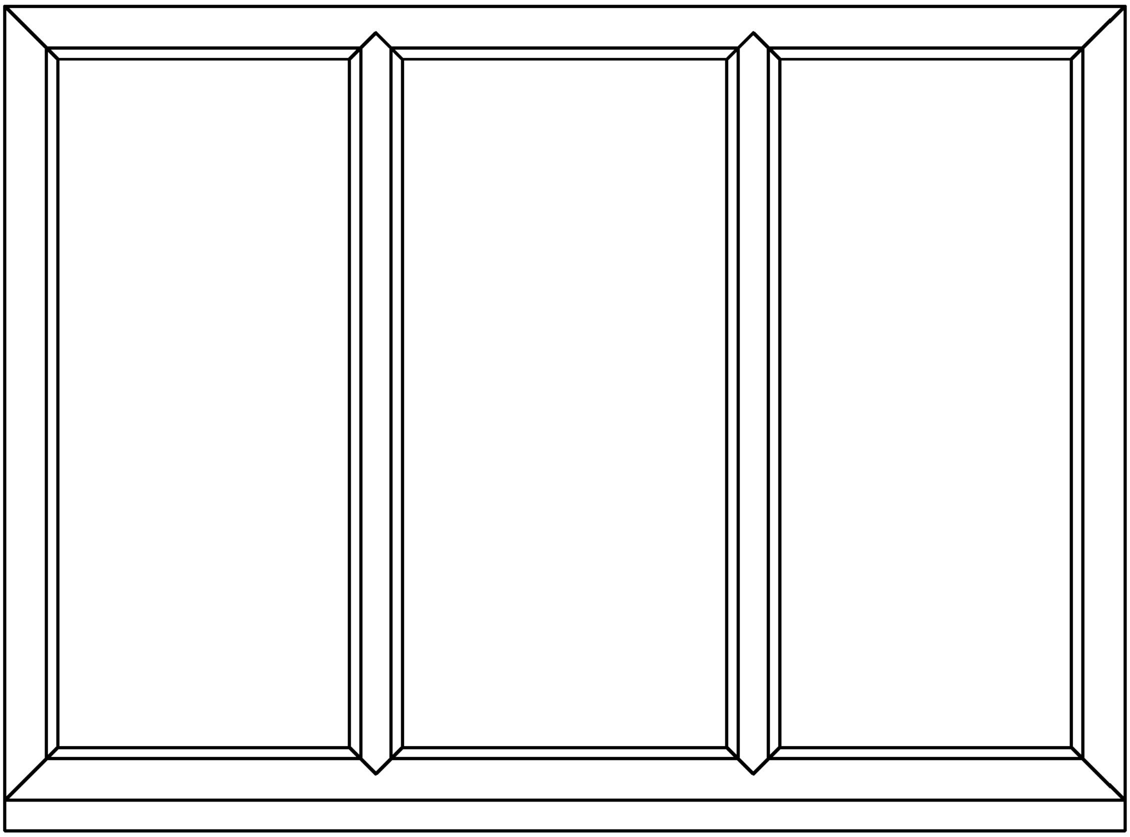

FIG. 7F

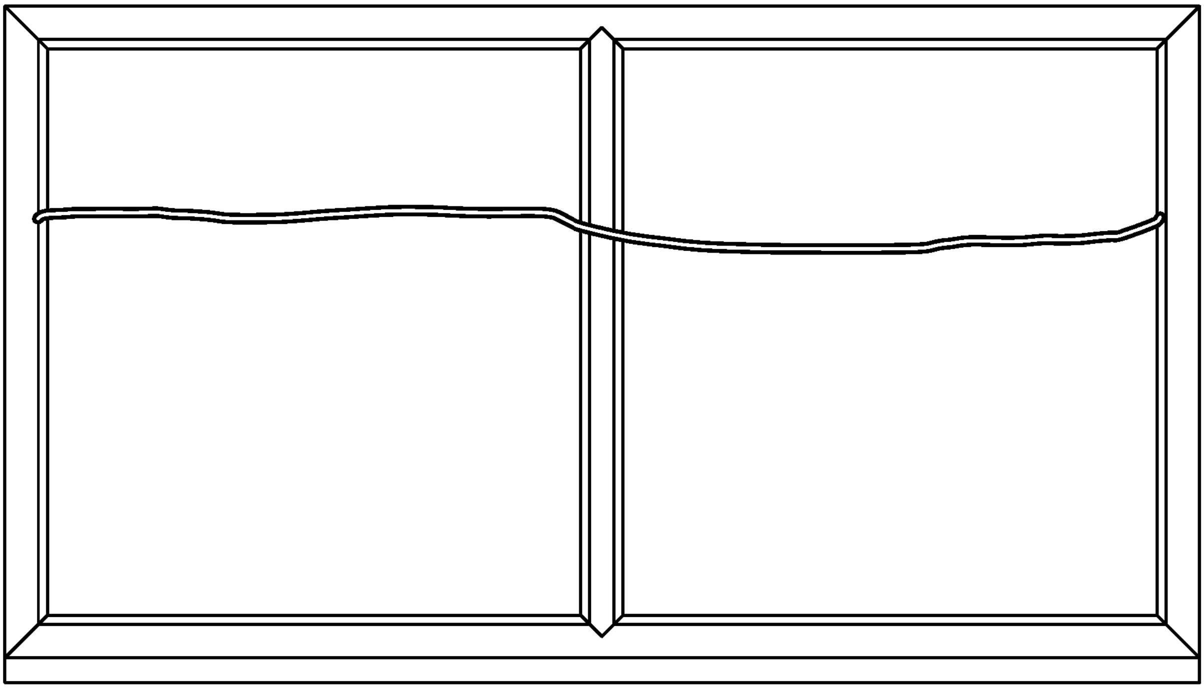

FIG. 8A

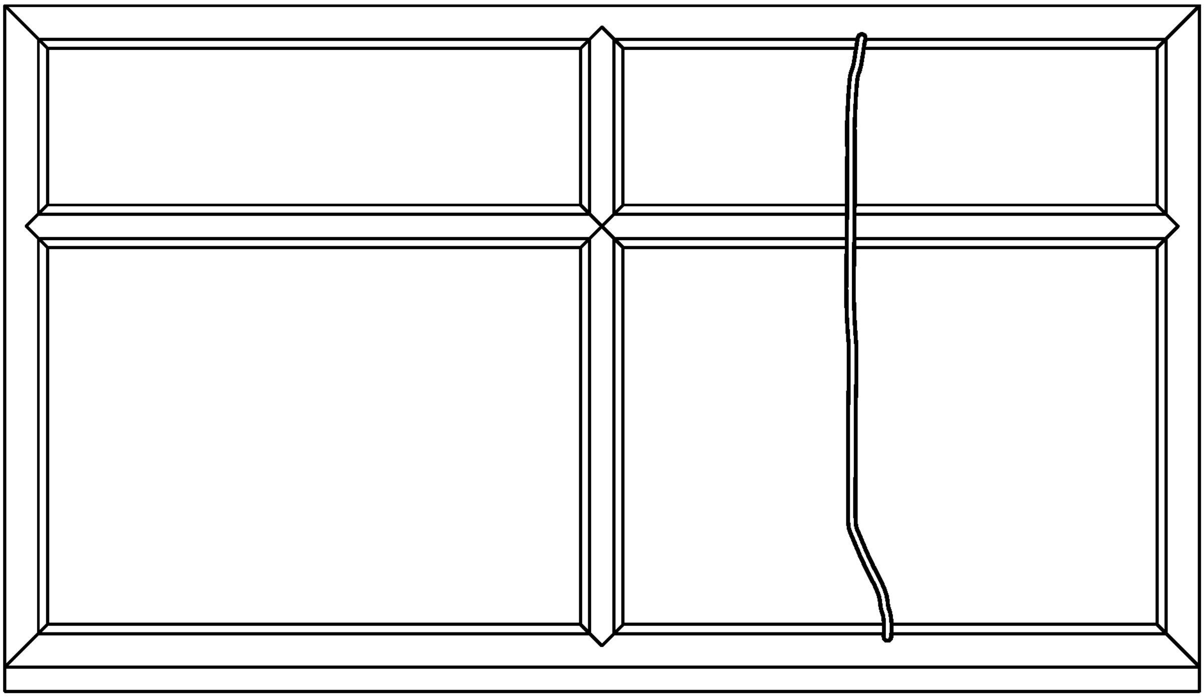

FIG. 8B

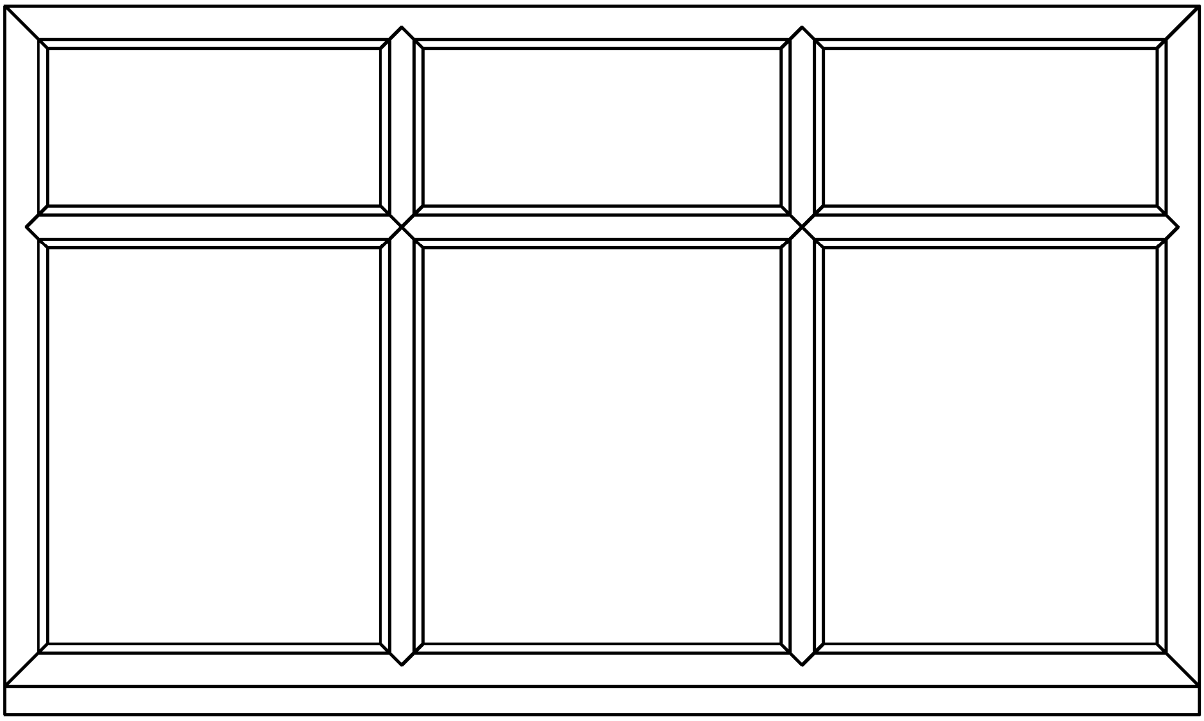

FIG. 8C

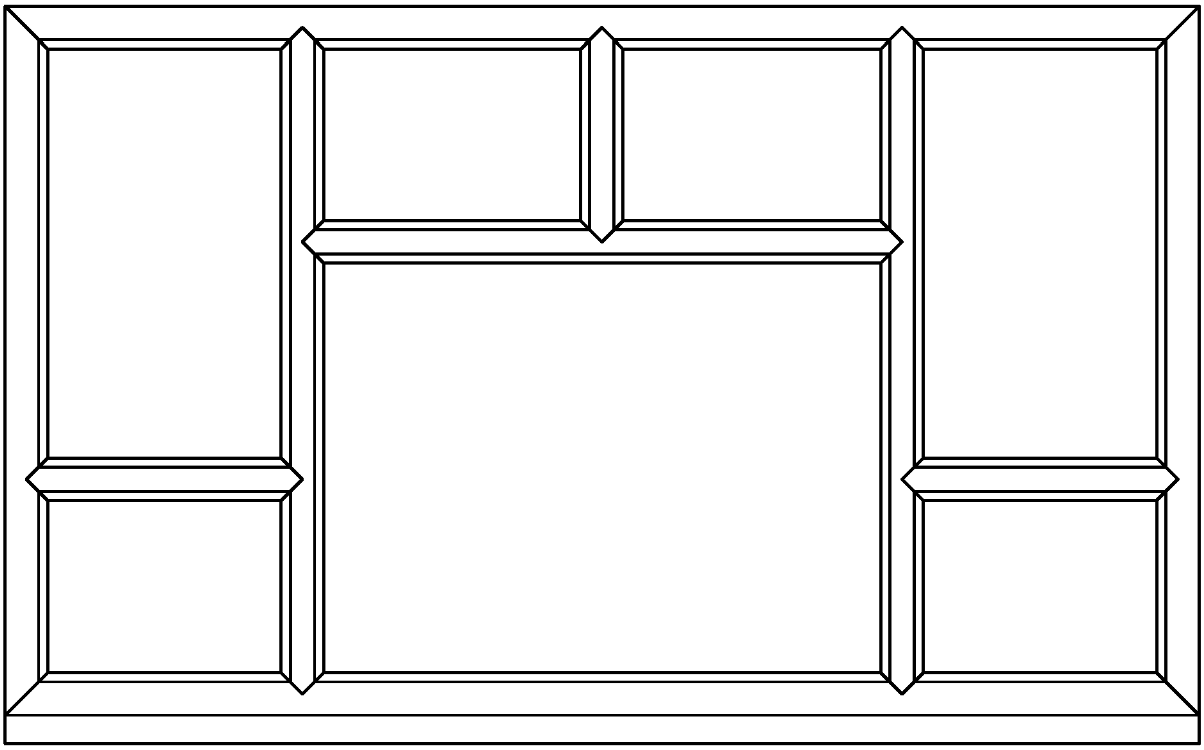

FIG. 9A

01 03 17

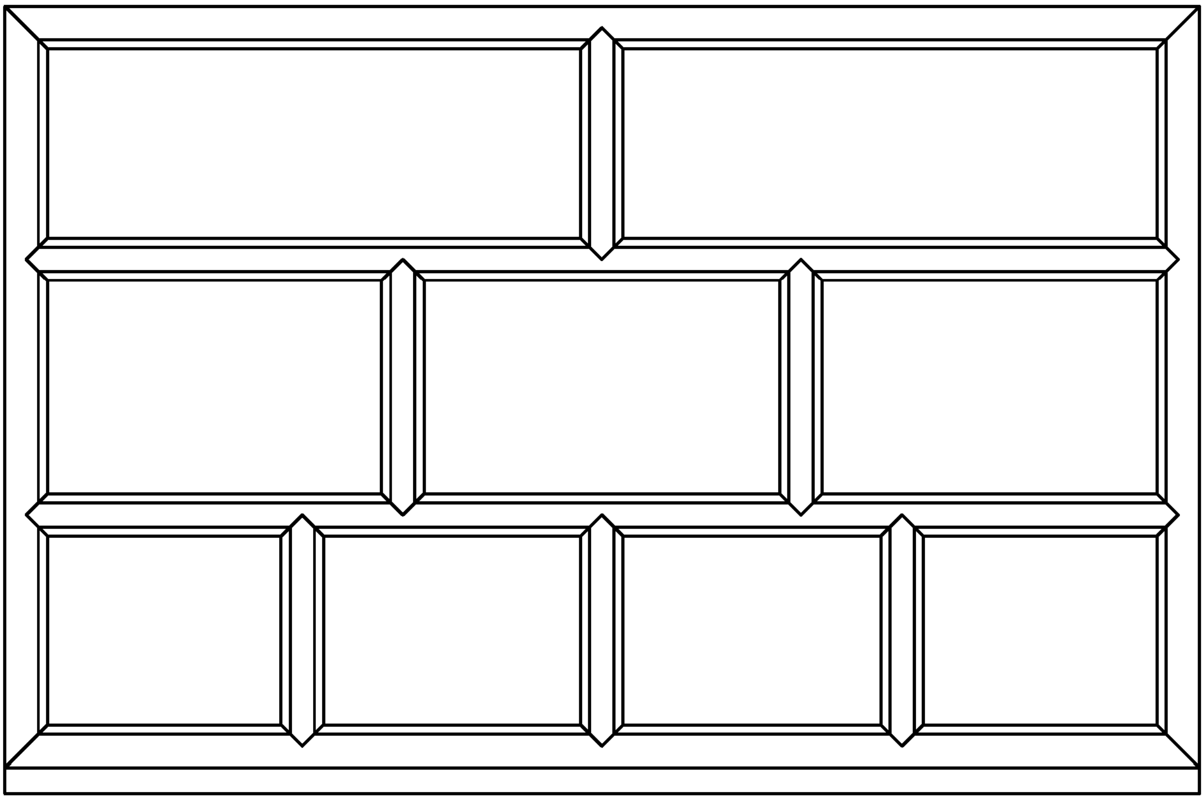

FIG. 9B

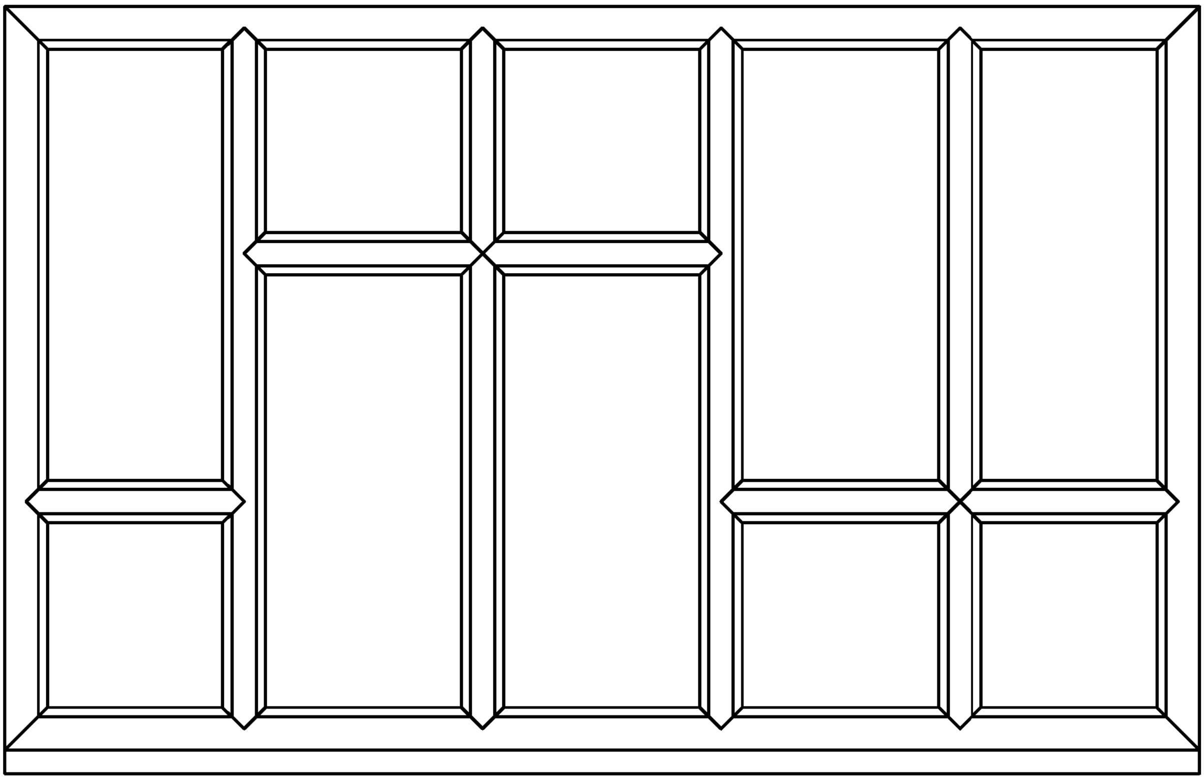

FIG. 9C

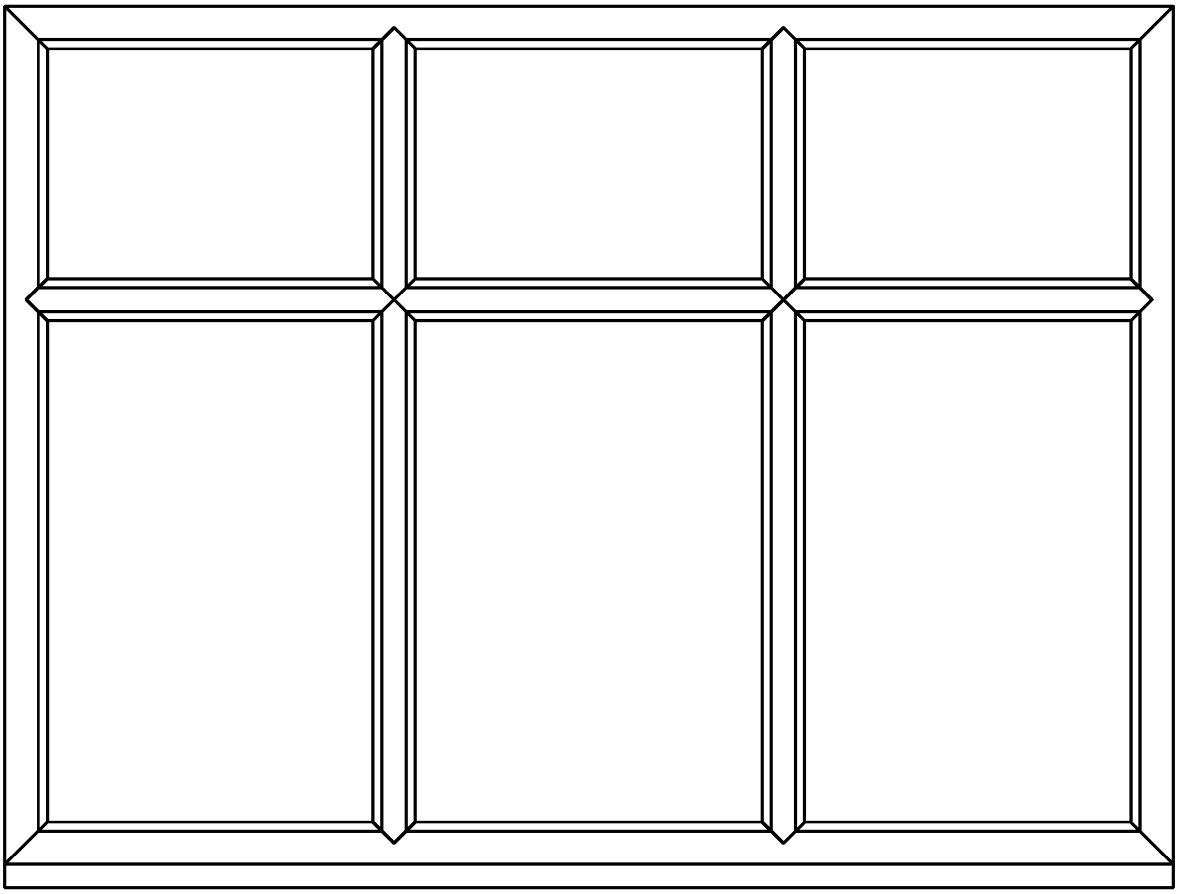

FIG. 10A

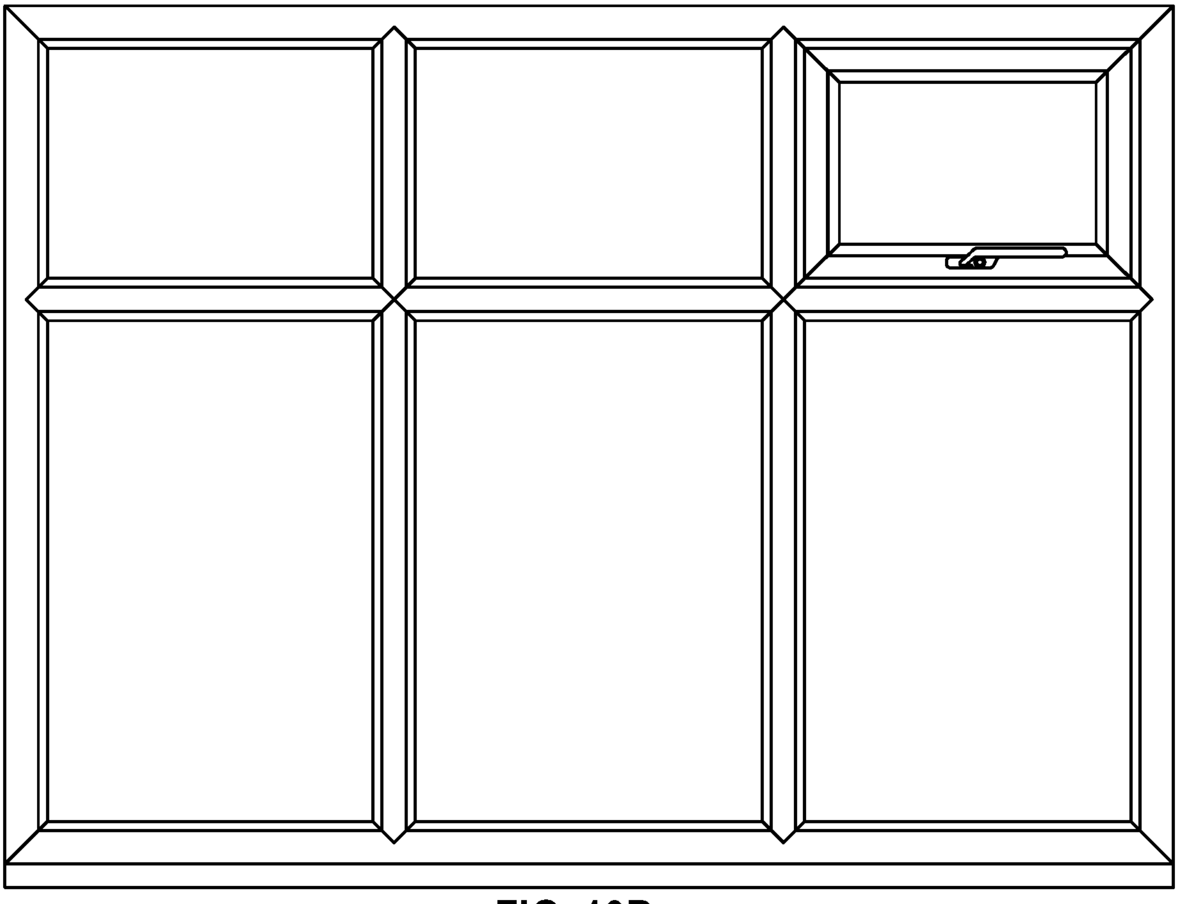

FIG. 10B

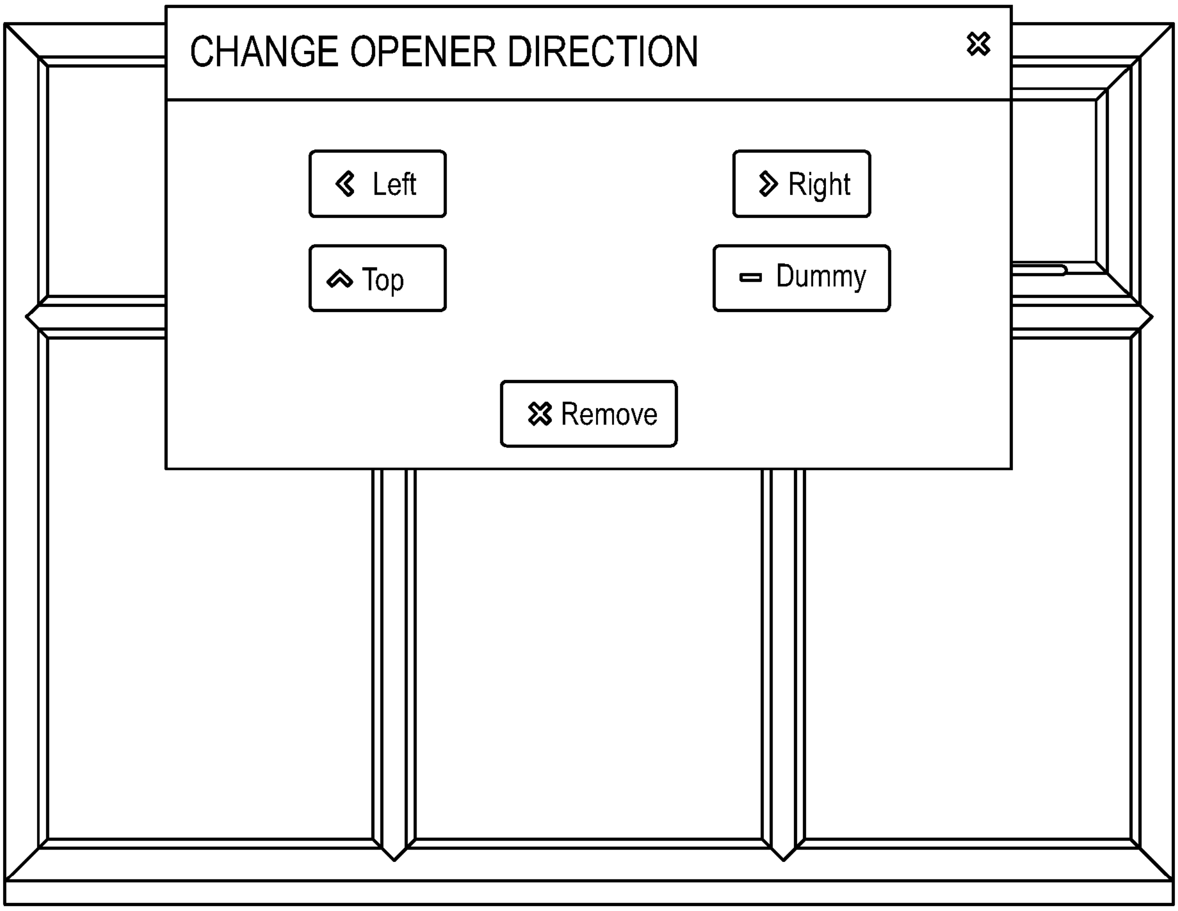

FIG. 10C

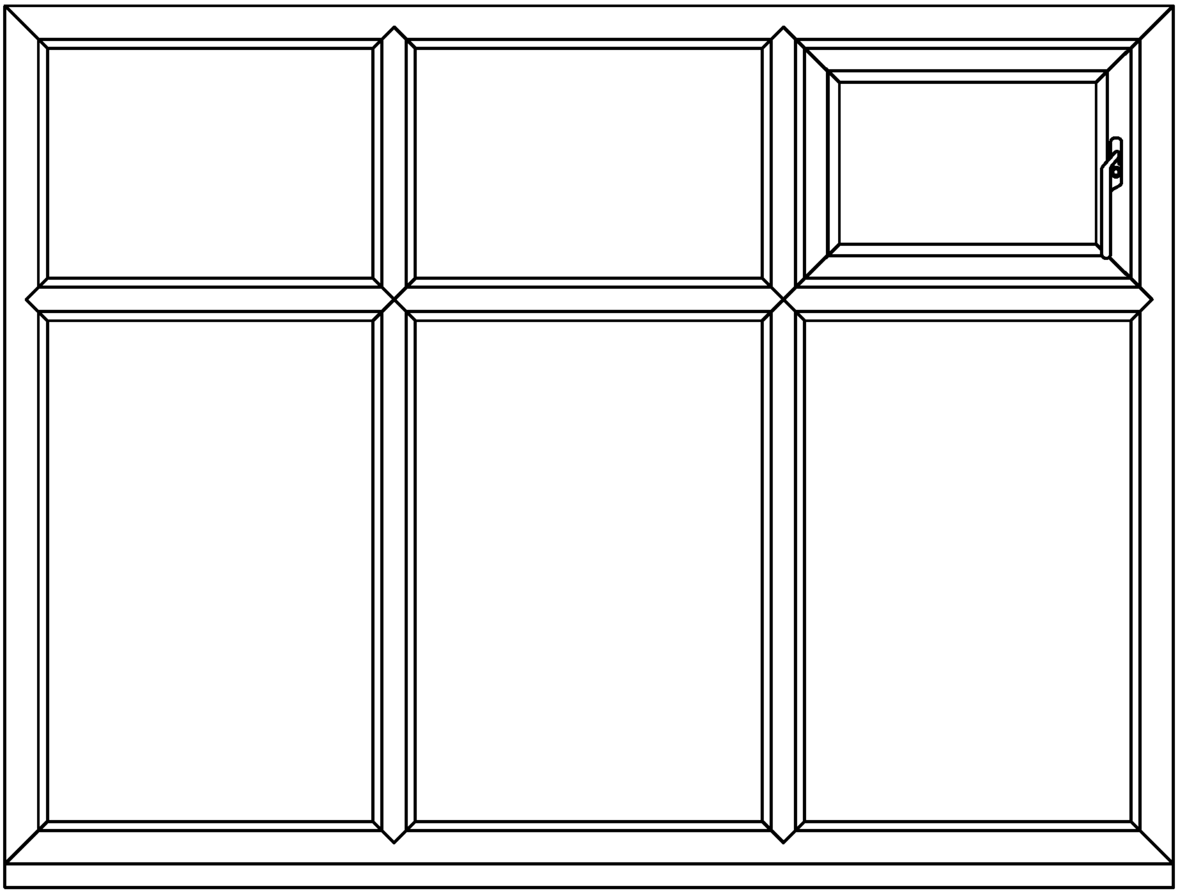

FIG. 10D

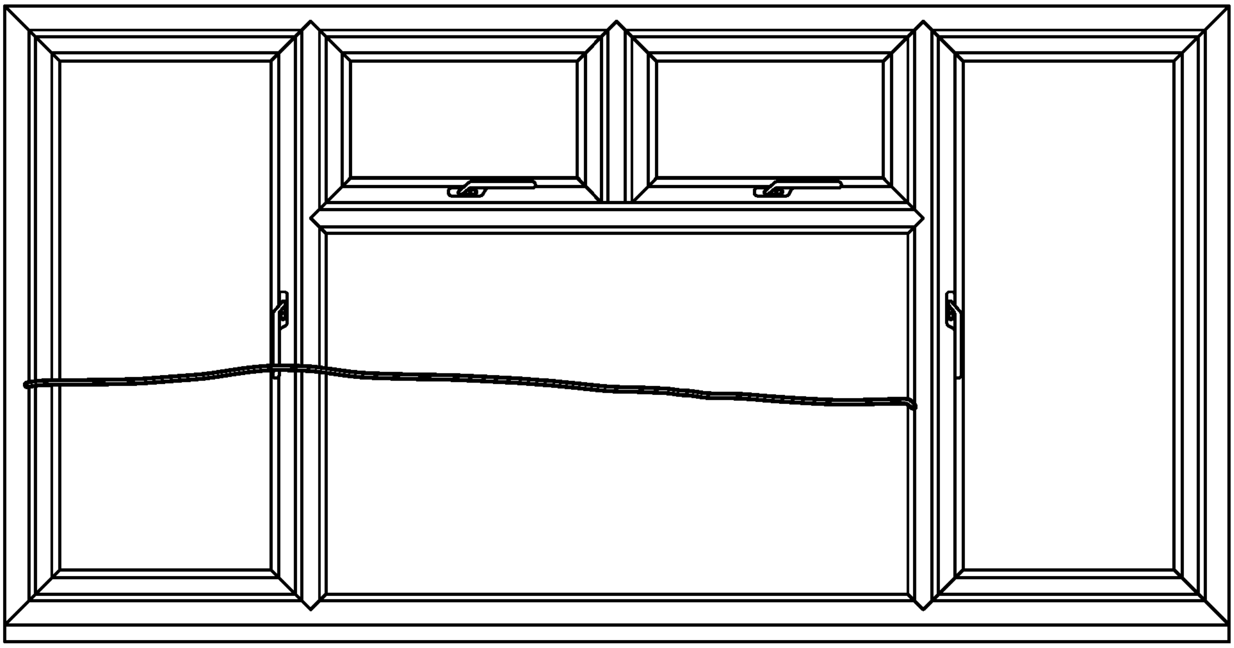

FIG. 11A

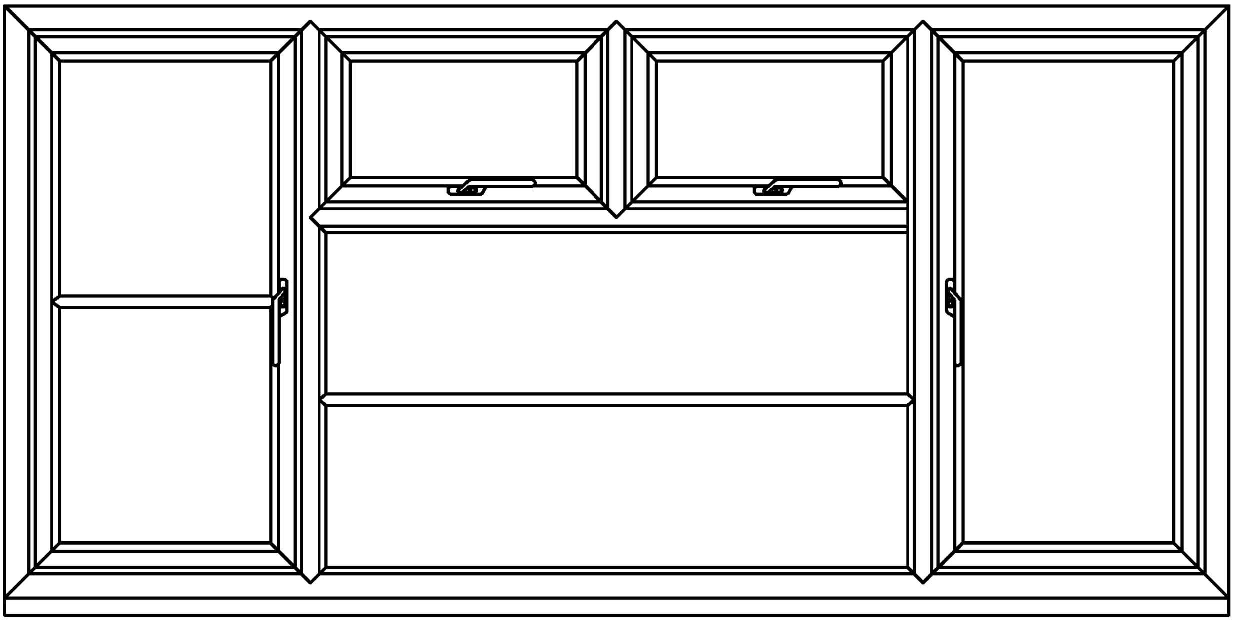

FIG. 11B

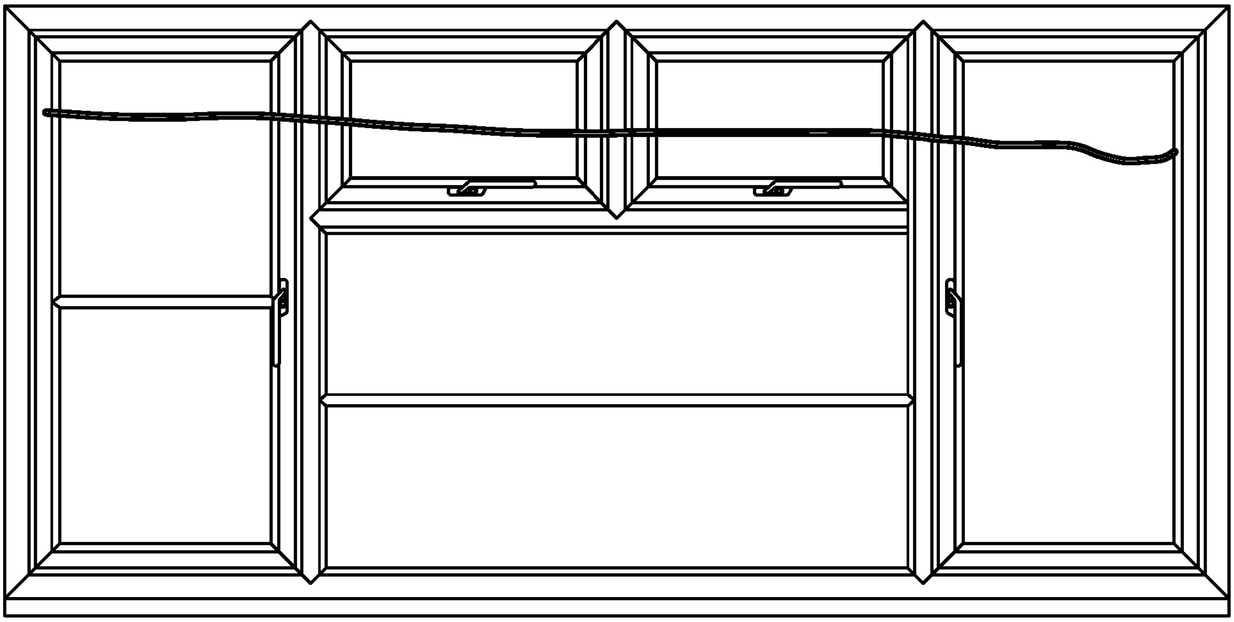

FIG. 11C

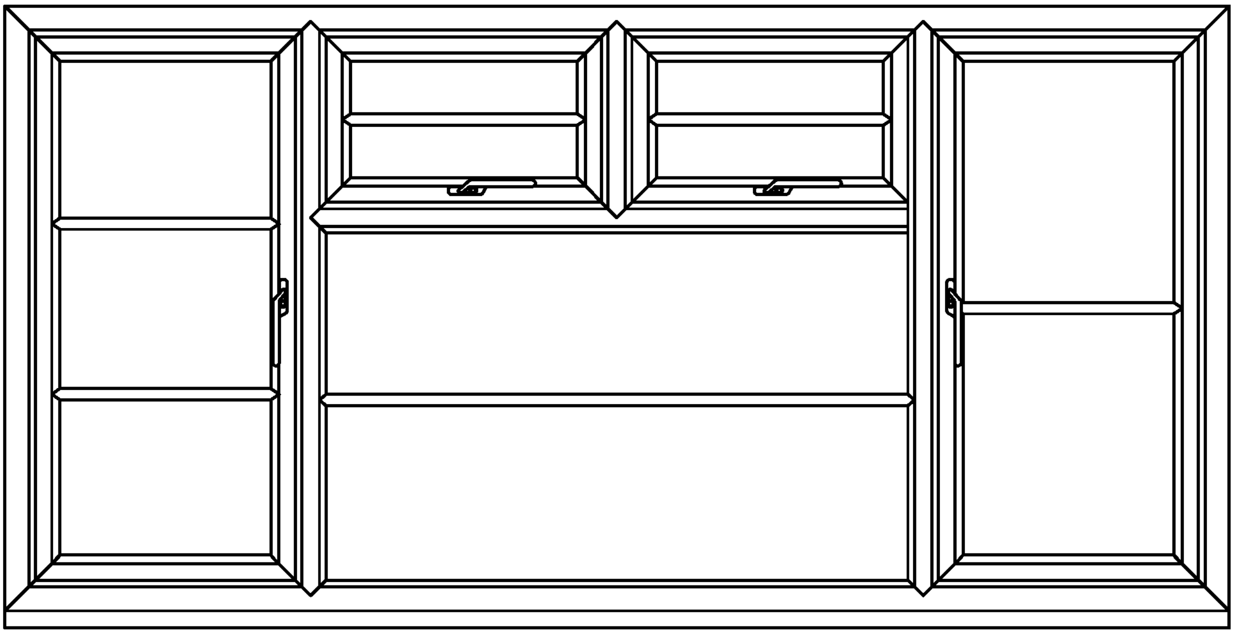

FIG. 11D

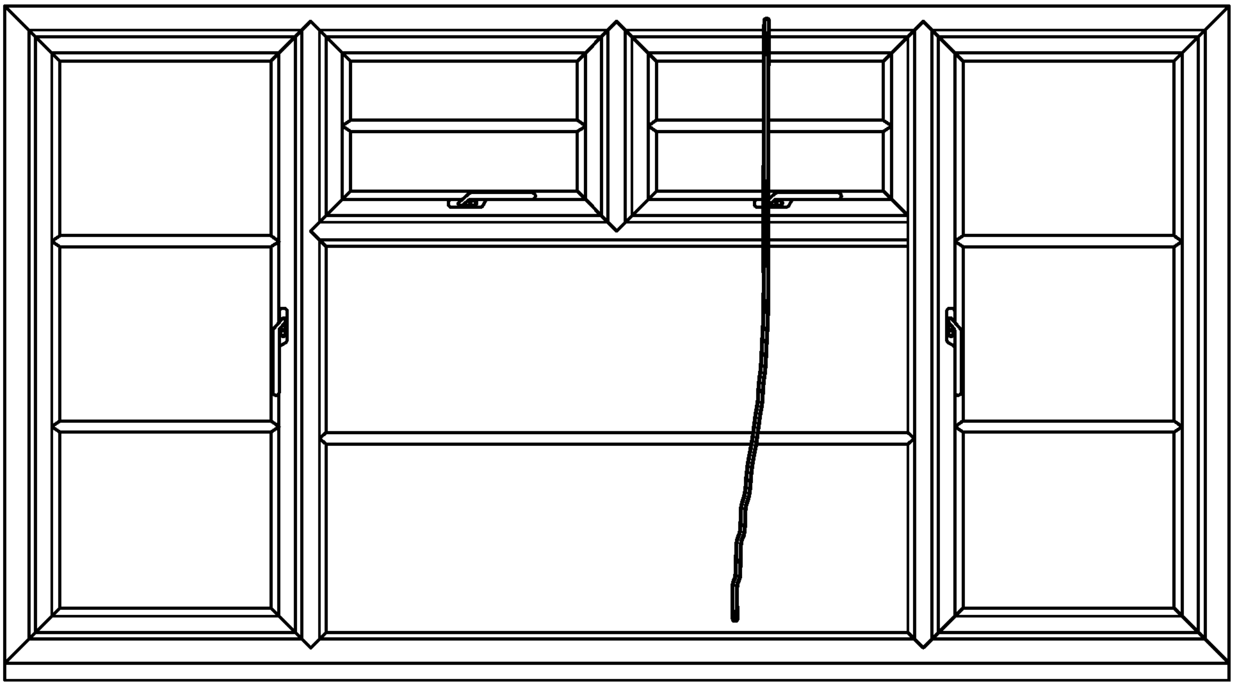

FIG. 11E

01 03 17

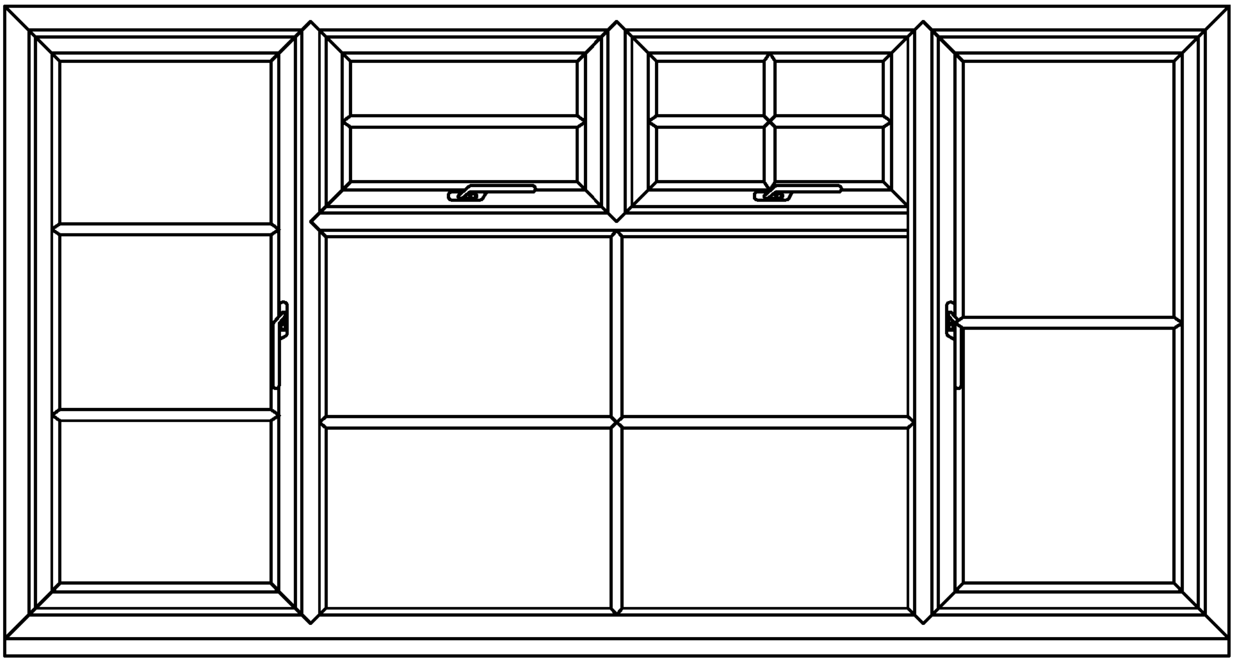

FIG. 11F

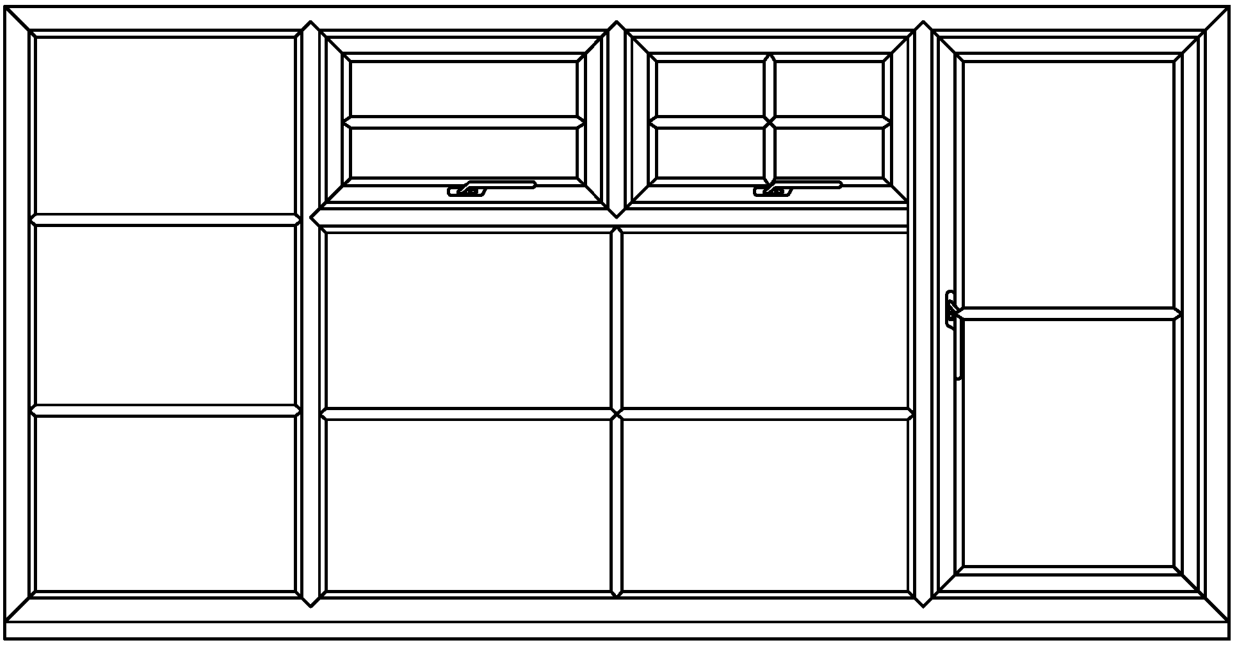

FIG. 12

01 03 17

A User Interface Mechanism

Field of Invention

The present invention is in the field of user interfaces. More particularly, but not exclusively, the present invention relates to augmented user interfaces for defining building frames.

Background

In the construction industry there is a desire for customisation from customers. These customisations can relate to building frames, such as window- and door-frames.

At present, and to facilitate the customisation of frames, one or more designs are presented on paper or electronically to a customer. The customer selects the design and indicates the changes they would like to have made. A designer or architect takes the customer's changes and works with a Computer-Aided Design (CAD) program to develop a frame incorporating the modifications proposed by the customer. The new design is printed for, or shared electronically with, the customer.

This process is relatively cumbersome as skilled designers/architects, typically operating remotely, are required to redesign the building frame for the customer. Furthermore, it is difficult for the customer to visualise the resulting customisation which could lead to numerous and time-consuming iterations.

Therefore, there is a desire for a user interface which enables less skilled individuals to rapidly create customised building frames.

It is an object of the present invention to provide a user interface mechanism which overcomes the disadvantages of the prior art, or at least provides a useful alternative.

Summary of Invention

According to a first aspect of the invention there is provided a computer-implemented method of defining window/door frames within a user interface, including:

a) receiving input from a user to select one of a plurality of window/door frame types, each window/door frame type associated with a set of manufacturing constraints;
b) receiving input from a user at an input interface to define a frame, wherein the input includes one or more discrete input paths;
c) detecting one or more geometric objects within the one or more discrete input paths using predefined thresholds
d) processing the detected geometric objects in relation to the associated set of manufacturing constraints to generate a frame comprising, at least, one or more mullions and one or more transoms; and
e) displaying the frame to the user.

The input interface may be a touch-screen interface. The frame may be displayed to the user on the touch-screen interface.

The input may be processed based upon context. The context may relate to a phase within a process; a touched location within a displayed frame; and/or a user-selectable mode.

The constraints may be selected from one of a plurality of constraints.

The constraints may include aesthetics constraints and/or manufacturing constraints.

The input interface may be at a user device, such as a tablet.

The input may be processed at a server. The constraints may be retrieved by the server from a database.

The input may be received from the user at a plurality of phases. Each phase is associated with specific constraints.

During processing, one or more geometric objects may be detected within the input. The classes of geometric objects may include shapes and lines. At least some of the phases may be associated with detecting a single class of geometric object. A geometric object may be detected when a beginning and/or ending of the input is within specific threshold of possible start and/or end-points for the geometric object. If the geometric object is a shape, the start and end-points may be defined as the same location; and if the geometric object is a line, the start and endpoints may be defined as being on an existing shape outline or line.

According to a further aspect of the invention there is a user device for defining window/door frames within a user interface, including:

an input apparatus configured to receive input from a user to select one of a plurality of window/door frame types, each window/door frame type associated with a set of manufacturing constraints, and to receive input from a user at an input interface to define a frame, wherein the input includes one or more discrete input paths;

a processor configured to detect one or more geometric objects within the one or more discrete input paths using predefined thresholds, and to process the detected geometric objects in relation to the associated set of manufacturing constraints to generate a frame comprising one or more mullions and one or more transoms; and

an output configured to display the generated frame to the user.

Other aspects of the invention are described within the claims.

Brief Description of the Drawings

Embodiments of the invention will now be described, by way of example only, with reference to the accompanying drawings in which:

Figure 1: shows a block diagram illustrating a device in accordance with an embodiment of the invention;

Figure 2: shows a block diagram illustrating an architecture of a device in accordance with an embodiment of the invention;

Figure 3: shows a flow diagram illustrating a method in accordance with an embodiment of the invention;

Figure 4: shows a flow diagram illustrating a method for an application in accordance with an embodiment of the invention;

Figure 5: shows a screenshot illustrating paths drawn on a user interface in accordance with an embodiment of the invention;

Figures 6a to 6b:

show screenshots illustrating the processing of "rectangular" path into a generated building frame within a user interface in accordance with an embodiment of the invention;

Figures 7a to 7f:

show screenshots illustrating the processing of "linear" paths into mullions for a building frame within a user interface in accordance with an embodiment of the invention;

Figures 8a to 8c:

show screenshots illustrating the processing of "linear" paths into transoms and mullions for a building frame within a user interface in accordance with an embodiment of the invention;

Figures 9a to 9c:

show screenshots illustrating different generated building frames within a user interface in accordance with an embodiment of the invention;

Figures 10a to 10d:

show screenshots illustrating the processing of input within a sashes mode into sashes for a building frame within a user interface in accordance with an embodiment of the invention;

Figures 11a to 11f:

show screenshots illustrating the processing of “linear” input within a glazing mode into glazing bars for a building frame within a user interface in accordance with an embodiment of the invention; and

Figure 12: shows a screenshot illustrating removal of a sash opening within a sashes mode within a user interface in accordance with an embodiment of the invention.

Detailed Description of Preferred Embodiments

The present invention provides a method and device to facilitate the definition of building frames within a user interface.

The inventor has discovered that construction constraints can be utilised within a user interface to process user input to generate viable building frames.

Such a user interface could be used by less skilled individuals such as salespeople or customers to both generate and visualise accurate and realistic building frames.

In Figure 1, a device 100 in accordance with an embodiment of the invention is shown.

The device 100 includes a processor 101, a display 102, an input apparatus 103, and a memory 104.

The device 100 may be a portable computing apparatus such as a smart-phone, tablet or smart-watch, or a laptop, or desktop computer.

The display 102 and input apparatus 103 may be unified in a combined input/display apparatus 105 such as a near-touch/touch-screen. Alternatively, the display 102 and input apparatus 103 may be separate, for example, where the input apparatus is a pointer device such as a mouse, or a near-touch/touch pad. In one embodiment, the input apparatus 105 functions with a digital stylus.

The memory 104 may be configured to store software applications 106, libraries 107, an operating system 108, and device drivers 109.

The processor 101 is configured to execute the software applications 106, libraries 107, operating system 108, and device drivers 109.

The software applications 106 may include an HTML5 enable web browser, such as Chrome®, Internet Explorer®, Safari®, Firefox®, or Opera™. In one embodiment, the web browser supports the Canvas® element and Javascript®.

The device is configured to perform the method described in relation to Figure 3. The device may be configured to execute a computer program to perform the method. The computer program may be stored in the memory 104. In one embodiment, the computer program is executed within a web browser executing on the device. In one embodiment, the computer program may be configured to interoperate with one or more servers to perform at least part of the method.

Referring to Figure 2, the various layers of the architecture 200 of the device 100 will be described.

Application software 201 (e.g. 106) is provided at a top layer. Below this layer are user interface APIs 202 which provide access for the application software 201 to user interface libraries. Below this layer are operating system APIs 203 which provide access for the application software 201 and user interface libraries to the core operating system 204. Below the core operating system

204 are the device drivers 205 which provide access to the input and display hardware.

Referring to Figure 3, a method 300 for providing a user interface mechanism on a device (such as 100) in accordance with an embodiment of the invention will be described.

In step 301, input is received from the user at a device (e.g. 100) to define the building frame. The input may be received via a touch-pad or touch-screen, or via a pointer mechanism. The input may consist of one or more discrete input paths from the input apparatus at the device 100. For example, the input may represent movement by a user within a 2D space, such as movement of a finger across a touch-screen.

In step 302, the input is processed in accordance with constraints to generate a building frame. In one embodiment, a user may select or define one of a plurality of building frame types before the processing step. A set of constraints may be associated with each of the types and retrieved either from local storage at the device 100 or from a server via a communication system for processing. In this embodiment, steps 301 to 303 may occur without the need for communication between the device 100 or the server.

In an alternative embodiment, the input is transmitted to a server via a communications system and the processing occurs at the server. The generated building frame may be transmitted from the server back to the device 100 for subsequent display to the user.

In one embodiment, the input is processed to detect geometric objects (such as shapes and lines). The geometric objects may be defined approximately within the input, for example, the input may include an approximately circular path from which a circle can be detected, the input may include an approximately rectangular path from which a rectangle can be detected, and/or the input may include an approximately linear path from which a line can be detected. In one embodiment, the linear paths may be detected to be horizontal or vertical. The approximations of the shapes or lines may be detected within predefined thresholds.

The geometric objects may be detected when a beginning and ending of the path within a specific threshold of possible starts and/or end-points for the geometric object. For example, where the geometric object is a shape, the end point may be predefined the beginning of the path and where the geometric object is a line, the start-points and end-points may relate to geometric objects already detected (i.e. the line must start and end near to a rectangular outline, or the line must start near a shape outline and end near another detected line).

Detected geometric objects may be utilised to generate the building frame within the constraints. For example, a detected approximate rectangle may be used to generate a rectangular frame if this is possible within the constraints and a detected approximate vertical line may be used to generate a mullion within the rectangular frame if this is possible within the constraints. The widths of the frame and mullions may also be defined by the constraints. The location of the mullion within the generated building frame may be symmetric within the frame. A detected approximate horizontal line may be used to generate a transom within the rectangular frame if this is possible within the constraints.

In one embodiment, the user interface mechanism may transition through different phases or operate in different modes. Different constraints may apply to different phases or modes. For example, in a first phase, only a single geometric shape may be detected in the input to contribute to generation of the building frame (i.e. the outside frame of the building frame), in a second phase, only lines may be detected in the input to contribute to generation of the building frame (i.e. mullions or transoms), and in one mode, only lines may be detected in the input to contribute to generation of the building frame (i.e. glazing bars).

In addition to constraints associated with each building frame type, each building frame type may be associated with one or more options. The options may be associated with the building frame type, with the different phases or modes, or with one or more components of the building frame. The user interface mechanism may then permit the selection of any of the one or more options in accordance with the building frame type, phase/mode, or component selected. In one embodiment, constraints associated with a building frame type, a phase/mode, or a component may restrict the options otherwise applicable for selection.

In step 303, the generated building frame is displayed on the device 100.

In one embodiment, the user input may be displayed to the user on the device 100 as input is provided. For example, a rough line may be displayed corresponding to user's input before steps 302 and 303 take place. The generated building frame may displace the rough line when displayed in step 303.

In one embodiment, the user interface mechanism may provide for alternative views of the generated building frame to be displayed. Alternative views may be displayed upon selection of an action by the user. In one embodiment, an alternative view may be the display of the generated building frame from the outside rather than inside. This may, for example, assist in the visualisation of the generated building frame by the user.

Embodiments of the present invention will now be described with reference to Figures 4 to 12.

These embodiments provide a method and system to enable a user to specify and configure a range of windows and doors in a customisable way from within a browser of their user device. The web application executing within the browser is configured to recognise/detect how a user has traced a path on the screen of their device. The application then generates a photorealistic graphic based on the size, position and shape that has been drawn and in accordance with constraints which may be defined for a specific product.

In these embodiments, the technology required to run the application is any machine running an HTML5 enabled web browser. This includes, for example, typical smart-phones, tablets, PCs, Macs and hybrids.

The browser could be any of the common types: Chrome®, Internet Explorer®, Safari®, Firefox®, Opera™. More specifically, in this embodiment, the browser is to support the Canvas® element (http://www.w3.org/TR/2009/WD-html5-20090825/the-canvas-element.html). In addition, in this embodiment, the browser is to have JavaScript® enabled.

The application also utilises the Fabric JavaScript® graphics library which is a collection of functions used to aid in the creation of graphics for the HTML5 canvas element and SVG shapes.

It will be appreciated by a person skilled in the art that alternative technologies to those identified above could be used in deploying alternative implementations of these embodiments and that the requirements specified above are exemplary only.

There is no installation process as the application is written in JavaScript® which is executed within the client's browser. The user navigates to a URL and is then authenticated with a username/password combination. This allows the application to load from the server only the part of the system that is applicable to the user type.

A product type is chosen (i.e. Window or Door) and the user is then presented with a blank graph paper background on the screen.

On a touch device the user can then trace a path which is rendered on the screen as it is traced (a mouse can be used on a non-touch device). Once the user stops tracing and, if the path is recognised, it is automatically replaced with a realistic graphical representation of the basic window shape which has been drawn, e.g. usually a rectangular frame.

The user then continues to trace more paths on top of the image to add further frame sections such as mullions and transoms. Each path drawn, if recognised, is positioned intelligently and symmetrically in the assumed correct place within the basic frame and rendered realistically. In this way the user can specify the configuration of the window/door by replicating the traditional method of using pen and paper. More advantageously the item is now captured digitally, rendered accurately and can be shown to others.

Once a user is satisfied with the style configuration of the item the user instructs the system to continue into a more advanced configuration mode.

The item is then centred on the screen, with default dimensions shown, along with a set of tabs which display the ‘configuration modes’ possible.

The user is in ‘Sashes mode’ by default and is invited to add any sash openers by touching (or clicking) any of the panes bounded by any of the mullions, transoms or frame.

A realistic graphical representation of the opener is then rendered in the correct position. The user can then touch any existing sash to change its orientation or to remove it.

At any time the user can choose either of the major dimensions shown (height and width) and change the overall size of the item. The image is then resized and rescaled appropriately according to the canvas size currently available to the user. The rescaling is applied precisely to all components of the item (e.g. thickness of frame, size of handle, etc) so that it retains its overall accurate proportional graphical representation.

The user continues to specify the full feature set and options for the item by selecting any of the configuration modes, for example: Sashes, Frame, Colour, Glass, Glazing, Hardware.

The application is intuitive in that it responds to where the user touches the item and, depending which mode is selected, presents a set of features and options to choose and change. As the user interacts the graphical representation and pricing is updated according to the choices made and any pre-configured values and rules.

There is a further ‘tracing’ function that is utilised when in ‘Glazing mode’. The user can add glazing bars positioned symmetrically in the panes (or sashes) by simply tracing horizontally or vertically within the appropriate region. There is also an eraser icon which removes all glazing bars when selected.

Modes can be entered and exited as often as is required to complete the full specification of the required item and to support any temporary changes of options.

An exemplary method for the application is shown in Figure 4.

An exemplary implementation for the application will now be described with reference to Figures 5 to 12.

Initialise the Web Page

Load Data Model

On a preceding web-page, the User has been presented with a list of Manufacturer’s products for selection.

When a product is selected, the route of the corresponding URL is used to send a request to the server.

The server responds with a JSON payload which consists of all the data necessary to configure the product.

An exemplary JSON payload is shown below:

```
{
  "product" : {
    "id": "0bf49d2c-0dac-45ab-9cd6-24b70176fed0",
    "metadata": {
      "archived": false,
      "created": 1415289062033,
      "company": {
        "id": "c33256a2-b46e-48dd-8f89-d4b56f56cdd4",
        "name": "Duraflex",
        "url":
"http://api.tommytrinder.com/v1/companies/c33256a2-b46e-48dd-
8f89-d4b56f56cdd4"
      },
      "description": "Product Description",
      "name": "New Product Name",
      "productType": {
        "id": 1,
        "name": "Sash Window",
        "slug": "sash_window"
      },
      "url":
"http://api.tommytrinder.com/v1/products/0bf49d2c-0dac-45ab-
9cd6-24b70176fed0"
    },
    "attributes": [
      {
        "id": "f195520d-0262-4fa3-99f8-623fa6202343",
        "name": "Material",
        "slug": "product_material",
        "value": "Timber"
      },
      {
        "id": "9683314c-2924-4805-b38e-fe4cbe97ffd2",
        "name": "Glazed",
        "slug": "product_glazed",
        "value": "External"
      },
      {
        "id": "57b61839-696f-4734-9a52-67d7bd4286d4",
        "name": "Frame Width",
        "slug": "product_frame_width",
        "value": 58
      }
    ],
    "components": [
    {
      "id": 29,
      "name": "Hardware",
      "slug": "hardware",
      "components": [
        {
          "id": "9f4aca16-a0b3-11e4-89d3-123b93f75cba",
          "componentType": {
```

```
          "id": 33,
          "name": "high security",
          "requiresDefault": true,
          "slug": "high_security"
        },
        "default": false,
        "name": "Component 1",
        "attributes": [
          {
            "id": "bedbf930-4235-4ce7-aec4-167371414a3d",
            "name": "Hardware Material",
            "slug": "hardware_material",
            "value": "brass"
          },
          {
            "id": "996346b5-d3cf-4095-8123-2e3268afe3a8",
            "name": "Hinge Stack Height In MM",
            "slug": "hinge_stack_height_in_mm",
            "value": 26
          },
          {
            "id": "996346b5-d3cf-4095-8123-2e3268afe3a8",
            "name": "Heavy Duty",
            "slug": "heavy_duty",
            "value": true
          }
        ],
        "prices": [
          {
            "id": "0dedb0fe-fd26-4efa-b4b4-6528e124d291",
            "name": "Price (per unit)",
            "slug": "price_per_unit",
            "value": 3.12
            },
            {
              "id": "0dedb0fe-fd26-4efa-b4b4-6528e124d292",
              "name": "Price (per m)",
              "slug": "price_per_m",
              "value": 2.74
            }
          ],
          "constraints": [
            {
              "id": "0dedb0fe-fd26-4efa-b4b4-6528e124d291",
              "name": "Min Length (mm)",
              "slug": "min_length_mm",
              "value": 35
            },
            {
              "id": "191eac58-8007-4066-a6e0-1a5013bba62e",
              "name": "Max Length (mm)",
              "slug": "max_length_mm",
              "value": 90
            }
          ],
          "url":
"http://api.tommytrinder.com/v1/components/9f4aca16-a0b3-11e4-
89d3-123b93f75cba"
        },
        {
          "id": "afab9980-a0b3-11e4-89d3-123b93f75cba",
          "componentType": {
            "id": 34,
            "name": "friction",
            "requiresDefault": true,
            "slug": "friction"
```

```
},
"default": true,
"name": "Component 2",
"attributes": [
  {
    "id": "bedbf930-4235-4ce7-aec4-167371414a3d",
    "name": "Hardware Material",
    "slug": "hardware_material",
    "value": "steel"
  },
  {
    "id": "996346b5-d3cf-4095-8123-2e3268afe3a8",
    "name": "Hinge Stack Height In MM",
    "slug": "hinge_stack_height_in_mm",
    "value": 16.5
  },
  {
    "id": "996346b5-d3cf-4095-8123-2e3268afe3a8",
    "name": "Heavy Duty",
    "slug": "heavy_duty",
    "value": false
  }
],
"prices": [
  {
    "id": "0dedb0fe-fd26-4efa-b4b4-6528e124d291",
    "name": "Price (per unit)",
    "slug": "price_per_unit",
    "value": 1.45
  },
  {
    "id": "0dedb0fe-fd26-4efa-b4b4-6528e124d292",
    "name": "Price (per m)",
              "slug": "price_per_m",
              "value": 3.67
            }
          ],
          "constraints": [
            {
              "id": "0dedb0fe-fd26-4efa-b4b4-6528e124d291",
              "name": "Min Length (mm)",
              "slug": "min_length_mm",
              "value": 40
            },
            {
              "id": "191eac58-8007-4066-a6e0-1a5013bba62e",
              "name": "Max Length (mm)",
              "slug": "max_length_mm",
              "value": 80
            }
          ],
          "url":
"http://api.tommytrinder.com/v1/components/afab9980-a0b3-11e4-
89d3-123b93f75cba"
        }
      ]
    },
    {
      "id": 12,
      "name": "Profile",
      "slug": "profile",
      "components": [
        {
          "id": "9f4aca16-a0b3-11e4-89d3-123b93f75cba",
          "componentType": {
            "id": 47,
  "name": "frame",
  "requiresDefault": true,
  "slug": "frame"
},
"default": true,
"name": "38mm Slim Frame",
"attributes": [
  {
    "id": "bedbf930-4235-4ce7-aec4-167371414a3d",
    "name": "Mould Shape",
    "slug": "mould_shape",
    "value": "ovalow"
  },
  {
    "id": "996346b5-d3cf-4095-8123-2e3268afe3a8",
    "name": "Mould Width In MM",
    "slug": "mould_width_in_mm",
    "value": 15
  },
  {
    "id": "996346b5-d3cf-4095-8123-2e3268afe3a8",
    "name": "Included In Base Price",
    "slug": "included_in_base_price",
    "value": true
  }
],
"prices": [
  {
    "id": "0dedb0fe-fd26-4efa-b4b4-6528e124d291",
    "name": "Price (per unit)",
    "slug": "price_per_unit",
    "value": 0
  },
            {
              "id": "0dedb0fe-fd26-4efa-b4b4-6528e124d292",
              "name": "Price (per m)",
              "slug": "price_per_m",
              "value": 3.12
            }
          ],
          "constraints": [
            {
              "id": "0dedb0fe-fd26-4efa-b4b4-6528e124d291",
              "name": "Min Length (mm)",
              "slug": "min_length_mm",
              "value": 35
            },
            {
              "id": "191eac58-8007-4066-a6e0-1a5013bba62e",
              "name": "Max Length (mm)",
              "slug": "max_length_mm",
              "value": 300
            }
          ],
          "url":
"http://api.tommytrinder.com/v1/components/9f4aca16-a0b3-11e4-
89d3-123b93f75cba"
        },
        {
          "id": "afab9980-a0b3-11e4-89d3-123b93f75cba",
          "componentType": {
            "id": 46,
            "name": "mullion",
            "requiresDefault": true,
            "slug": "mullion"
          },
"default": true,
"name": "Z Mullion",
"attributes": [
  {
    "id": "bedbf930-4235-4ce7-aec4-167371414a3d",
    "name": "Mould Shape",
    "slug": "mould_shape",
    "value": "flat"
  },
  {
    "id": "996346b5-d3cf-4095-8123-2e3268afe3a8",
    "name": "Mould Width In MM",
    "slug": "mould_width_in_mm",
    "value": 12
  },
  {
    "id": "996346b5-d3cf-4095-8123-2e3268afe3a8",
    "name": "Included In Base Price",
    "slug": "included_in_base_price",
    "value": true
  }
],
"prices": [
  {
    "id": "0dedb0fe-fd26-4efa-b4b4-6528e124d291",
    "name": "Price (per unit)",
    "slug": "price_per_unit",
    "value": 0
  },
  {
    "id": "0dedb0fe-fd26-4efa-b4b4-6528e124d292",
    "name": "Price (per m)",
    "slug": "price_per_m",
              "value": 2.78
            }
          ],
          "constraints": [
            {
              "id": "0dedb0fe-fd26-4efa-b4b4-6528e124d291",
              "name": "Min Length (mm)",
              "slug": "min_length_mm",
              "value": 25
            },
            {
              "id": "191eac58-8007-4066-a6e0-1a5013bba62e",
              "name": "Max Length (mm)",
              "slug": "max_length_mm",
              "value": 200
            }
          ],
          "url":
"http://api.tommytrinder.com/v1/components/afab9980-a0b3-11e4-
89d3-123b93f75cba"
        }
      ]
    }
  ]
  }
  "message": "Product has been successfully loaded"
}
```

The application assigns this data to model for the product.

Initialise Appropriate Scaling

The **scaling** is used to represent actual **mm** in equivalent screen **pixels** (px).

Firstly, the **viewport** *width* and *height* available for current device (measured in px) are detected by the application.

Then a scaling value is initialised, dependent on the viewport width, for example, as follows:

```
  var maxWidth = 3000;    /* -- maximum initial width of the
item in mm -- */
  defaultScaler = Math.round(viewportWidth / maxWidth * 1000)
/ 1000;
  if (defaultScaler > 1)    {defaultScaler = 1;}
  else if (defaultScaler < 0.3)    {defaultScaler = 0.3;}
```

In the above example, an actual maximum width of 3000mm is set.

Thus scaling defaults to the viewportWidth divided by 3000, rounded to 3 decimal places.

The maximum scale is then limited to 1 and the minimum limited to 0.3: a typical viewport width of 1,068px giving a scaling of 0.356.

Thus in the example: 0.356px represents 1mm, or 1000px represents ~2,809mm.

Similarly, an actual frame width of 58mm may be represented on the screen by 58 * 0.356 = 20.648px.

Page Layout

The application defines a **header** and **footer**, each 60px high, positioned at the top and bottom of the screen in the browser displayed on the user's device.

The header may be used for a retailer's logo at top left, and some navigational links on the right.

The footer is used for various control objects, as well as buttons for saving and cancelling.

The rest of the space is initialized by the application as a **canvas** area.

A background image is loaded to tile the entire canvas, for example: a blue graph paper. Each major square on the graph paper image measures 50px x 50px. Each minor square on the graph paper image measures 10px x 10px.

Initialise Canvas Drawing Mode

The application uses FabricJS, an open source library, to provide an interactive object model for the HTML5 <canvas> element.

The application uses this library to intialise the canvas for drawing mode:

```
var canvas = new fabric.Canvas('itemCanvas', {
    isDrawingMode: true
});
```

The application can set some appropriate options for the cursor style, brush colour and brush width:

```
  function doCanvasDrawingSettings(c)   {
      // -- set default drawing settings --
      c.freeDrawingCursor = 'crosshair';
      c.freeDrawingBrush.color = '#F2F2F2'; // -- very light
gray --
      c.freeDrawingBrush.width = 5;
}
```

Initial Path Trace

Tracing Paths

The input device can be either a mouse, finger or digital pen.

The user traces a path with the input device and the application can then analyse this path.

Each path starts and ends when the user starts and stops tracing.

The example in Figure 5 shows two distinct paths drawn by the user.

W3C Standard Syntax

W3C standards define an Open Web Platform for application development to enable developers to build rich interactive experiences.

The syntax described below follows the W3C standards for SVG paths which all browsers implement.

The path information is stored as a sequence of coordinates and corresponding "command" tokens eg:
M 100,200 L 200,400 L 300,200 z corresponds to a triangle with vertices (100,100) (300,100) (200,100).

Note that the origin (0,0) is at the top/left corner.

The M indicates a *moveto*, the L indicate *lineto*, and the z indicates a *closepath.*

In these embodiments, the user is drawing freehand which can be defined as

a sequence of quadratic Bézier segments, eg:
M,0,0,Q,2.5,1,8,2,Q,13.5,3,21,3.5,Q,28.5,4,35,4.5,Q,41.5,5,45.5,5,Q,49.5,5,51,5,L,52.5,5

Analysing Path Data

The application uses the following event listener to store information about each path traced:

```
canvas.on('path:created', function(pth) {
    // -- path information is stored here in pth --
}
```

The application then analyses the initial drawn path for the following properties:

1 The width and height of the bounding box of the path;

2 The start and end points; and

3 The general shape of the path.

The application then applies the following rules for the drawing of an initial basic frame, such that:

- The width and height must both be greater than the threshold of 100px. A value of 100px has been selected after testing. 100px is an optimal value that ignores any mistaken path traces but supports users who may be using a small device such as a phone. It will be appreciated that different threshold values could be used, for example, to support different device sizes or input modalities.
- Both pairs of start point and end point ordinates must be within the threshold of 100px of each other. This ensures that the user is intending to finish where they started, (i.e. to complete the drawing of a frame). The nominal value of 100px allows for a natural margin of error whilst still enforcing completeness. It will be appreciated that the threshold value is exemplary and similar threshold values may be used,

or alternative threshold values to support other devices/inputs may be used.

- Most window and door products are framed with a rectangle as the initial shape. However, some products may support circular frames and recognition may be required when an approximate circle has been attempted.

Below shows exemplary code for testing properties 1 and 2.

```
var threshold = 100;
if (Math.abs(mpath.x.start - mpath.x.end) < threshold){
    if (Math.abs(mpath.y.start - mpath.y.end) < threshold){
        if (mpath.width > threshold && mpath.height >
threshold)    {
            // -- we have a basic frame drawn --
            canvas.sections.basicFrame = true;
        }
    }
}
```

Recognising The Shape

There follows an explanation of the path analysis that allows the application to recognise if the user was attempting to draw a rectangle, circle or neither.

Here is a typical sample of the sequence of quadratic Bézier curve segments for a drawn path:

character	**CONTROL x**	**CONTROLy**	**END x**	**END y**
Q	46.5	0	47	0
Q	47.5	0	47.25	0.5
Q	47	1	47	3.5

Q	47	6	47	9
Q	47	12	46.5	16.5
Q	46	21	45	25
Q	44	29	43.5	33.5
Q	43	38	42	43.5
Q	41	49	40.5	57.5
Q	40	66	38.5	76.5
Q	37	87	35.5	98.5
Q	...	...	...	...

The first pair of ordinates signify the *control point* of the curve and the second pair define the *end point* the path passes through.

For analysis by the application, it *ignores the control points* and simply looks at the *set of end points* the path actually passes through.

First, the application considers the *change in x* (dx) and the *change in y* (dy) of each end point compared to the previous point

char	**CONTROLx**	**CONTROLy**	**END x**	**END y**	**change in ENDx (dx)**	**change in ENDy (dy)**
Q	46.5	0	47	0	-	-
Q	47.5	0	47.25	0.5	0.25	0.50
Q	47	1	47	3.5	-0.25	3.00
Q	47	6	47	9	0.00	5.50
Q	47	12	46.5	16.5	-0.50	7.50
Q	46	21	45	25	-1.50	8.50
Q	44	29	43.5	33.5	-1.50	8.50
Q	43	38	42	43.5	-1.50	10.00
Q	41	49	40.5	57.5	-1.50	14.00
Q	40	66	38.5	76.5	-2.00	19.00

Q	37	87	35.5	98.5	-3.00	22.00
Q	34	110	32.5	120.5	-3.00	22.00
Q	31	131	29.5	140	-3.00	19.50
Q	28	149	26.5	158	-3.00	18.00
Q	...	...	...	...	...	...

Rectangle or Circle?

For a rectangle, the *changes in the x ordinate will remain fairly small when the changes in y are large*, and vice versa. This would signify that the user is drawing a *roughly horizontal or vertical line.* For a circle the changes in x ordinate will be *much more similar in magnitude to the changes in y ordinate.* This is because for curved paths both ordinates are *generally changing together.* The application, therefore, analyses the dx and dy pairs and counts **what percentage of them have either the absolute value of dx or dy being less than a nominal value of 3?**

For well-drawn rectangles, the inventor has found that *over 80%* conform to this test, whereas for well-drawn circles *less than 20%* conform.

After extensive testing, the inventor has found that, to allow quite 'wobbly' rectangles and circles, the decision breakpoint should be set at *50%.*

We than also test on the final criteria as follows:

The changes in direction

The application can also analyse the *magnitude* and *polarity** of the dx and dy pairs, as follows:

For a rectangular path the 4 lines will be traced, such that:

- dx is stongly positive whilst dy stays fairly constant (right)

- dy is strongly positive whilst dx stays fairly constant (down)
- dx is strongly negative whilst dy stays fairly constant (left)
- dy is strongly negative whilst dx stays fairly constant (up)

Note these 4 could be combined in 8 different ways, depending on how the user has drawn the rectangle, ie:

- Clockwise: 1,2,3,4 or 2,3,4,1 or 3,4,1,2 or 4,1,2,3
- Anti-Clockwise:4,3,2,1 or 3,2,1,4 or 2,1,4,3 or 1,4,3,2

For an approximate circle the trace can be seen as 4 arcs, such that:

- dx is fairly positive whilst dy is fairly positive (right & down)
- dx is fairly negative whilst dy is fairly positive (left & down)
- dx is fairly negative whilst dy is fairly negative (left & up)
- dx is fairly positive whilst dy is fairly positive (right & up)

Again, as above, the same 8 sequences would be valid examples of how a user might draw a circle.

To eliminate irregular shapes being recognised the application ensures that one of these two path patterns are valid.

If all these criteria are met, then a rectangle or circle is recognised. The application removes the visible path trace and use its bounding box co-ordinates to render a realistic graphical representation of a window.

An exemplary user's path trace is shown in Figure 6a and the graphical representation of the basic frame for that trace is shown in Figure 6b.

Rendering Basic Frame

Utilising Data Model Default Properties

The application uses the default properties set in the manufacturer's product data model which was loaded when page was initialized.

For the basic frame window the following default properties may be set:

1 Frame Colour
2 Joint Type
3 Top, Bottom, Left, Right Frame Widths
4 Bead Width
5 Bead Shape
6 Cill Height
7 Glass Pattern

The values for the above properties allow the application to create and position a set of objects on the canvas.

Some sample code for drawing the *left vertical frame section*

```
function doLeftVerticalFrame(c,p) {

    // -- initialise array for objects that will be grouped --
    var objsForGroup = [];

    // -- left vertical --
    p.id = "leftVerticalFrame";
    p.left = p.transX;
    p.top = p.transY;
    p.points = [
        {x: 0, y: p.fh},
        {x: 0, y: 0},
        {x: p.fr, y: p.bfr},
        {x: p.fr, y: p.fh - p.bfr}
    ];
    p.img_frame = p.img_vt_frame;                   // -- for
pattern fill --
    p.shadow = p.viewFromInside ? true : false;     // -- set
```

```
object shadow --
    // -- add section to group --
    objsForGroup.push(doSection(p));

    p.id = "leftVerticalInner";
    p.left = p.transX + p.frnotbd;
    p.top = p.transY + p.frnotbd;
    p.points = [
        {x: p.frnotbd, y: p.fh - p.frnotbd},
        {x: p.frnotbd, y: p.frnotbd},
        {x: p.fr, y: p.bfr},
        {x: p.fr, y: p.fh - p.bfr}
    ];
    // -- add bead polygon object to group --
    objsForGroup.push(doBead(p));

    p.id = "leftVerticalBeadLine";
    p.x1 = p.transX + p.frnotbd;
    p.y1 = p.transY + p.frnotbd;
    p.x2 = p.transX + p.frnotbd;
    p.y2 = p.transY + p.fh - p.bfrnotbd;
    p.beadLineColour = p.viewFromInside ?
p.lightBeadLineColour : p.darkBeadLineColour;
    // -- add bead line object to group --
    objsForGroup.push(doBeadLine(p));

    // -- do rebate --
    if (p.viewFromInside)   {
        p.x1 -= 1;
        p.y1 -= 1;
        // -- shorten rebate to start at transom when we have
sash --
        if (c.sections.rightTransom)  {
            p.y1 = p.transY + p.tdrop + p.tfrnotbd;
        }
        p.x2 -= 1;
        p.y2 += 1;
        // -- add rebate line object to group --
        objsForGroup.push(doRebateLine(p));

        p.x1 -= p.rbt;
        p.y1 -= p.rbt;
        p.x2 -= p.rbt;
        p.y2 += p.rbt;
        // -- add rebate line object to group --
        objsForGroup.push(doRebateLine(p));

    }

    // -- get section length --
    var length = Math.round(p.fh / p.scaler * 100) / 100;

    // -- now create group --
    var group = new fabric.Group(objsForGroup, {
        name: 'leftVerticalFrameSection',
        category: 'frame',
        length: length,

        left: p.transX,
        top: p.transY,
        opacity: 0,
        hasControls: false,
        hasBorders: false,
        lockRotation: true,
        lockMovementX: true,
        lockMovementY: true,
            selectable: true
        });

        // -- add group to collection --
        c.objsToAdd.push(group);

    }
```

Continue Designing The Product

Vertical Lines – Mullions

Once the basic frame is loaded and the application invites the user to continue tracing paths with the input device.

For window products, the user may want to add one or more vertical lines, corresponding to mullions.

Again, the application captures each complete path trace for analysis and, if certain conditions are met (outlined below), the canvas is reloaded with the new graphic corresponding to a photorealistic graphical interpretation.

The first condition is that any new trace must start and end within a nominal tolerance of 50px of the basic frame. It will be appreciated that alternative tolerances may be used for different devices or input methods.

This ensures that the path does not stray too much outside the frame borders, thus eliminating any mistaken traces and also guiding the user to be quite precise in what they want to draw.

The application may use the following *isPathBoxed()* function, shown below:

```
/**
```

```
 * checks that path is within bounding box of basic frame
 * @param {Object} bb - bounding box of basic frame
 * @param {float} bb.top - vertical distance from top of
canvas to top of frame bounding box
 * @param {float} bb.bottom - vertical distance from top of
canvas to bottom of frame bounding box
 * @param {float} bb.left - horizontal distance from left of
canvas to left of frame bounding box
 * @param {float} bb.right - horizontal distance from left of
canvas to right of frame bounding box
 * @param {Object} pbb - bounding box of path
 * @param {float} pbb.top - vertical distance from top of
canvas to top of path bounding box
 * @param {float} pbb.bottom - vertical distance from top of
canvas to bottom of path bounding box
 * @param {float} pbb.left - horizontal distance from left of
canvas to left of path bounding box
 * @param {float} pbb.right - horizontal distance from left of
canvas to right of path bounding box
 * @returns {Boolean} true or false
 */
 function isPathBoxed(bb, pbb)   {

    // -- tolerance to draw outside of frame --
    var tolerance = 50;

    // -- check that path is within bounding box of basic
frame --
    if (pbb.left > bb.left - tolerance)  {   // -- path left
within tolerance of bb left
        if (pbb.right < bb.right + tolerance)  {    // -- path
right within tolerance of bb right
            if (pbb.top > bb.top - tolerance)  {     // --
path top within tolerance of bb top
                if (pbb.bottom < bb.bottom + tolerance)  { //
-- path bottom within tolerance of bb bottom
                    return true;
                }
            }
        }
    }
    return false;
}
```

If the path has strayed too much as shown in Figure 7a it is simply removed so that the user can try again.

The next condition is to test for the path being a minimum length and “fairly” vertical.

For this, the application may use the function *isPathVerticalLine()* (described below) which uses **18°** as the maximum angle the path can stray from the vertical and **50px** as the minimum vertical length of the path.

Again these are nominal values that can be adjusted/modified for different products.

With user testing, it has been discovered that **18°** is the optimal maximum angle for this context, as being neither too strict nor too ambiguous.

Similarly, the minimum length of **50px** usefully eliminates mistaken traces.

Exemplary code for the *isPathVerticalLine()* function is given below:

```
/**
 * checks that path is fairly vertical
 * @param {Object} pbb - bounding box of path
 * @param {float} pbb.width -  width of path bounding box
 * @param {float} pbb.height - height of path bounding box
 * @returns {Boolean} true or false
 */
function isPathVerticalLine(pbb)   {
    var minHeight = 50; // -- smallest possible height --
    // -- has a height and is within 18 degrees (arctan 1/3)
of vertical --
    if (pbb.height > minHeight && pbb.height > 3 * pbb.width){
        return true;
    }
    return false;
}
```

The application also tests for the path reaching both the top and bottom of frame, (i.e. fully spanning the basic frame).

For this the application may use the function *isPathVerticalLineFull()* (described below) which ensures that the top of path starts less than **20%**, and bottom of path ends more than **80%** of height of basic frame.

These nominal values were arrived at after user testing.

It is not necessary to force the user to be too accurate, but what the user is drawing cannot be ambiguous.

However, it will be appreciated that alternative values, such as 10% and 90% could be used to enforce more accuracy.

Exemplary code for the *isPathVerticalLineFull()* function is given below:

```
/**
 * checks that path reaches top and bottom of frame
 * @param {Object} bb - bounding box of basic frame
 * @param {float} bb.top - vertical distance from top of
canvas to top of frame bounding box
 * @param {float} bb.height - height of frame bounding box
 * @param {Object} pbb - bounding box of path
 * @param {float} pbb.top - vertical distance from top of
canvas to top of path bounding box
 * @param {float} pbb.bottom - vertical distance from top of
canvas to bottom of path bounding box
 * @returns {Boolean} true or false
 */
function isPathVerticalLineFull(bb, pbb)   {
    // -- top of path starts less than 20%, and bottom of path
ends more than 80% of height of basic frame --
    return (pbb.top - bb.top < bb.height / 5 && pbb.bottom -
bb.top > 4 * bb.height / 5);
}
```

An exemplary path that is successfully boxed and fulfils the 3 vertical line tests explained above is shown in Figure 7b.

The application can now replace the successful path with a **mullion** positioned so it *divides the glass area in half.*

The width of the mullion section itself and its various other properties are read from the manufacturer's product data model.

An example of a mullion positioned from the path shown in Figure 7b so it divides the glass area in half is shown in Figure 7c.

If any further paths are traced which also fulfil the *isPathVerticalLine()* then the application can simply add another *mullion* so that the *glass is always divided in equal widths* **no matter where the vertical line was drawn.**

For example, if the path was drawn to the right of the first mullion (as shown in Figure 7d), or drawn to the left of the first mullion (as shown in Figure 7e), it will be replaced with equidistant mullions as shown in Figure 7f.

Note that **in each redraw the complete item is reloaded from scratch** so that any pre-existing sections' positions can be adjusted.

The user can continue drawing as many full mullions as desired and the application will always position them equally within the basic frame.

Horizontal Lines – Transoms

At any point the user could also trace a horizontal line which corresponds to a transom.

In the same way as for mullions, similar conditions exist within the application to test that the trace is boxed, spans the full width and is fairly horizontal.

The transom(s) can be traced before or after the mullion(s).

Processing of paths corresponding to transoms will now be described with reference to a product where the constraints include two transoms positioned either 1/3 from top or 1/3 from bottom as a default. These constraints are driven from the manufacturer's product data model and can of course differ. For example, *cottage casement windows* never have transoms, so the path trace would simply be removed by the application if the user tried to draw one.

A user drawn path for a transom after a first mullion is shown in Figure 8a. The resulting generated building frame showing the transom and a user drawn path for a second mullion is shown in Figure 8b. This results in two mullions and a transom as shown in Figure 8c.

It will be appreciated that the application can support a range of window design layouts and that the layouts shown and described herein are exemplary.

The possible designs are read from the manufacturer's product data, and the *application will not allow the user to draw layouts that cannot be manufactured. That is, the product data defines constraints that apply when processing the user input to generate the building frames for visualisation.*

This is facilitated by labelling each of the designs and then mapping these labels to the manufacturer's labelling system.

Furthermore the possible design layouts for any product can be grouped into *standard* and *non-standard* which may affect the pricing. This information is also read from the data model.

Advanced Design Layouts

The application can continue to analyse path traces over what has been already represented graphically.

The rules outlined above can, for example, preserve symmetry whilst being able to support numerous possible designs. Examples of advanced design layouts supported by the application are shown in Figures 9a to 9c.

Design Completed

At any point in the above process the user can submit the design within the application as being **completed.**

This will allow the user to move on to configure all of the other window product details, ie Sashes, Frame, Colours, Glass, Glazing and Hardware options.

Configuring The Product

The user is presented with tabs by the application, each of which represent a different *mode* to help complete the configuration process.

The modes currently consist of:

- Sashes
- Frame
- Colours
- Glass
- Glazing
- Hardware, but can vary for each product type

The user can now touch or click on various sections of the product and the response will depend on which mode is currently selected, as explained in more detail below.

Sashes

The default mode after the design has been submitted as completed is *Sashes.*

If the user touches or clicks on any pane section of the product a Sash (or Opener) is instantly added to that pane.

The sizes of the sash sections are determined from the product data model.

The default direction of the opener is determined and dependent on which pane was clicked on.

For example, if a *top hung* pane is added then the sash is set to *open from the top* as the default.

At any time whilst in Sashes mode the user can touch or click on the sash again to reconfigure it's opener direction, or remove the opener completely.

An initial design is shown in “Sashes Mode” in Figure 10a. In Figure 10b, a top hung opener has been added to this design, opening to the top as a default. The top hung opener can have its opening direction changed as shown in Figure 10c. In Figure 10d, the top hung opener is now opening to the left.

Glazing

In *Glazing* mode the user is invited by the application to draw paths on the product to represent where the glazing bars (or astragals) should be located.

As before, the application analyses the path trace segments to interpret what is being drawn.

The application firstly uses the same criteria as above to decide if a line is mainly vertical or mainly horizontal (or neither):

- Using *isPathVerticalLineBoxed()* to determine if path is mainly vertical
- Using *isPathHorizontalLineBoxed()* to determine if path is mainly horizontal
- If neither of the above, then path is removed and no action is taken.

The application then analyses the bounding box of the path to determine which pane(s) the glazing bars are being drawn across by:

- Looping through every pane in the product and (if it is a mainly vertical line); and
- Running the function *isVerticalLineInsidePane()* to determine if the path starts, ends or passes through the pane.

Exemplary code for the *isVerticalLineInsidePanel()* function is shown below:

```
/**
```

```
 * To test if path starts, ends or passes through a pane
 * @param {Object} pbb - bounding box of path
 * @param {float} pbb.top - vertical distance from top of
canvas to top of path bounding box
 * @param {float} pbb.bottom - vertical distance from top of
canvas to bottom of path bounding box
 * @param {float} pbb.left - horizontal distance from left of
canvas to left of path bounding box
 * @param {float} pbb.right - horizontal distance from left of
canvas to right of path bounding box
 * @param {Object} pane object
 * @param {float} pane.width - width of pane
 * @param {float} pane.height - height of pane
 * @param {float} pane.left - horizontal distance from left of
canvas to left edge of pane
 * @param {float} pane.right - horizontal distance from left
of canvas to right edge of pane
 * @returns {Boolean} true or false
 */
function isVerticalLineInsidePane(pbb, pane)   {

    // -- path left and right inside pane top and bottom --
    if (pbb.left > pane.left && pbb.right < pane.left +
pane.width) {
        if (
        // -- EITHER path starts in top half of pane --
        (pbb.top > pane.top && pbb.top < pane.top +
pane.height / 2)
        ||
        // -- OR path ends in bottom half of pane --
        (pbb.bottom > pane.top + pane.height / 2 && pbb.bottom
< pane.top + pane.height)
        ||
        // -- OR middle of path is in pane --
        (pbb.top < pane.top && pbb.bottom > pane.top +
pane.height)
        )   {
            // -- SO if path only starts and ends in pane --
            if (pbb.top > pane.top && pbb.bottom < pane.top +
pane.height) {
                // -- THEN top of path must start at less than
20% from top of pane --
                // -- AND bottom of path must end at more than
80% of height of pane --
                if (!(pbb.top - pane.top < pane.height / 5
                    && (pbb.bottom - pane.top) > 4 *
pane.height / 5))  {
                    return false;
                }
            }
            return true;
        }
    }
    return false;
}
```

The application then adds the glazing bar to the pane, symmetrically placed as to any others that have already been added.

The width of the glazing bar is determined from the product data model.

In this way the user can intelligently add a wide variety of glazing bar placements, for example:

In Glazing mode, the first horizontally drawn path spans two panes as shown Figure 11a. This results in the symmetric addition of glazing bars as shown in

Figure 11b, taking account of any sashes already present. Another horizontally drawn path spans four panes as shown in Figure 11c, and the glazing bars are symmetrically added in the same way as shown in Figure 11d. A vertically drawn path spans two panes as shown in Figure 11e, and the glazing bars are symmetrically added again as shown in Figure 11f.

Mode Switching

The modes are fully independent so a user can freely switch between them.

For example, the user has switched back to Sashes mode to remove the left hand opener in Figure 11f and the glazing bars are now drawn for no opener in the left hand pane as shown in Figure 12.

Saving Data

Any changes made to the product configuration may be automatically saved to the server.

This allows the user to return later and reload a product they are working on.

The user can also copy the current product to use as the starting point for a similar product they want to design and configure.

Pricing Mechanism

The price of each component may be delivered in the product data model as set by the manufacturer.

Each component added and configured may have an impact on the pricing.

The component pricing can handle a combination of price per unit and price per length or area.

For every design or configuration change the total product price may be recalculated and presented to the user.

Pricing rules may be implemented where certain combinations of components infer a pricing uplift.

A potential advantage of some embodiments of the present invention is that, by processing input in accordance with stored constraints, building frames can be defined by individuals without specific construction or architectural knowledge.

While the present invention has been illustrated by the description of the embodiments thereof, and while the embodiments have been described in considerable detail, it is not the intention of the applicant to restrict or in any way limit the scope of the appended claims to such detail. Additional advantages and modifications will readily appear to those skilled in the art. Therefore, the invention in its broader aspects is not limited to the specific details, representative apparatus and method, and illustrative examples shown and described. Accordingly, departures may be made from such details without departure from the spirit or scope of applicant's general inventive concept.

Claims

1. A computer-implemented method of defining window/door frames within a user interface, including:
   a) receiving input from a user to select one of a plurality of window/door frame types, each window/door frame type associated with a set of manufacturing constraints;
   b) receiving input from a user at an input interface to define a frame, wherein the input includes one or more discrete input paths;
   c) detecting one or more geometric objects within the one or more discrete input paths using predefined thresholds
   d) processing the detected geometric objects in relation to the associated set of manufacturing constraints to generate a frame comprising, at least, one or more mullions and one or more transoms; and
   e) displaying the frame to the user.

2. A method as claimed in claim 1, wherein the input interface is a touch-screen interface.

3. A method as claimed in claim 2, wherein the frame is displayed to the user on the touch-screen interface.

4. A method as claimed in any one of the preceding claims, wherein geometric objects are detected and processed based upon a context associated with when a discrete input path within the input was received from the user.

5. A method as claimed in claim 4, wherein a user progresses through a series of input phases, the context relates to which phase the user is within, and wherein input is received from the user during each of the input phases.

6. A method as claimed in claim 4, wherein the context relates to a touched location within a displayed frame.

7. A method as claimed in claim 4, wherein the context relates to a user-selectable mode.

8. A method as claimed in any one of the preceding claims, wherein the input interface is at a user device.

9. A method as claimed in claim 8, wherein the user device is a tablet.

10. A method as claimed in any one of the preceding claims, wherein the input is processed at a server.

11. A method as claimed in claim 10, wherein the constraints are retrieved by the server from a database.

12. A method as claimed in any one of the preceding claims, wherein the input is received from the user at each of a plurality of input phases.

13. A method as claimed in claim 12, wherein each input phase is associated with specific constraints within the associated set of manufacturing constraints.

14. A user device for defining window/door frames within a user interface, including:

An input apparatus configured to receive input from a user to select one of a plurality of window/door frame types, each window/door frame type associated with a set of manufacturing constraints, and to receive input from a user at an input interface to define a frame, wherein the input includes one or more discrete input paths;

A processor configured to detect one or more geometric objects within the one or more discrete input paths using predefined thresholds, and to process the detected geometric objects in relation to the associated set of manufacturing constraints to generate a frame comprising one or more mullions and one or more transoms; and

An output configured to display the generated frame to the user.

15. A computer program configured, when executed on a device comprising an input apparatus, a processor, and an output, to perform the method of any one of claims 1 to 12.

16. An electronically readable medium configured to store the computer program of claim 15.