Canvas viewport mapping math involves converting coordinates between the "user space" (the logical coordinate system for your virtual content) and the "canvas space" (pixels on the HTML canvas). The key is to handle translation and scaling transformations so only the visible viewport portion of your virtual content is drawn on the canvas.

Here's a summary of the core math and concepts:

1. Viewport Definition
The viewport is a rectangle inside the virtual coordinate space representing what's currently visible.

Defined by:

viewport.x, viewport.y: top-left coordinate of the viewport in user space.

viewport.width, viewport.height: size of the viewport.

2. Scaling Factors
Compute ratios to map user space size to canvas pixel size:

scaleX = canvas.width / viewport.width

scaleY = canvas.height / viewport.height

3. Mapping a Point from User Space to Canvas Space
Given a point (pt.x, pt.y) in user space, convert it to canvas coordinates as:

c
a
n
v
a
s
X
=
(
p
t
.
x
−
v
i
e

w
p
o
r
t
.
x
)
×
s
c
a
l
e
X
canvasX=(pt.x−viewport.x)×scaleX
c
a
n
v
a
s
Y
=
(
p
t
.
y
−
v
i
e
w
p
o
r
t
.
y
)
×
s
c

a
l
e
Y
canvasY=(pt.y−viewport.y)×scaleY
This accounts for viewport offset and scales to fit the canvas pixels.

4. Using context.setTransform
Instead of manually calculating every point, you can set the canvas context transformation matrix directly:

```
js
context.setTransform(
  scaleX, 0,     // scaleX and skewX
  0,     scaleY, // skewY and scaleY
  -viewport.x * scaleX, // translateX
  -viewport.y * scaleY  // translateY
);
```

This matrix scales and translates drawing operations so drawing in user space can be done directly without manual conversions.

5. Inverse Mapping (Canvas to User Space)
To map mouse or pointer events from canvas coordinates back to user space:

u
s
e
r
X
=
c
a
n
v
a
s
X
s
c
a
l
e
X
+

v
i
e
w
p
o
r
t
.
x
userX=
scaleX
canvasX
+viewport.x
u
s
e
r
Y
=
c
a
n
v
a
s
Y
s
c
a
l
e
Y
+
v
i
e
w
p
o
r
t
.
y

userY=
scaleY
canvasY
 +viewport.y
Conceptual Example Class (TypeScript-like pseudocode)
ts
```
class Viewport {
  x: number; // top-left x of viewport in user space
  y: number; // top-left y
  width: number;
  height: number;
  canvasWidth: number;
  canvasHeight: number;

  get scaleX() { return this.canvasWidth / this.width; }
  get scaleY() { return this.canvasHeight / this.height; }

  pointToCanvas(pt: {x:number,y:number}) {
    return {
      x: (pt.x - this.x) * this.scaleX,
      y: (pt.y - this.y) * this.scaleY
    };
  }

  canvasToPoint(pt: {x:number,y:number}) {
    return {
      x: pt.x / this.scaleX + this.x,
      y: pt.y / this.scaleY + this.y
    };
  }

  applyTransform(context: CanvasRenderingContext2D) {
    context.setTransform(
      this.scaleX, 0,
      0, this.scaleY,
      -this.x * this.scaleX,
      -this.y * this.scaleY
    );
  }
}
```
Summary
Translation shifts the viewport origin to (0,0) in canvas space.

Scaling maps user space units to pixel coordinates.

Use setTransform on the canvas context for efficient drawing without manual math per point.

Inverse math allows mapping input events like mouse clicks back to your virtual coordinate system.

This method underlies high-performance canvas rendering techniques including virtualized tables, maps, and games.

Reference example and detailed explanation:
https://gedge.ca/blog/2024-11-03-virtualizing-the-canvas/.

To transform user (mouse or pointer) coordinates to canvas coordinates, especially when the canvas has been translated, scaled, or otherwise transformed with the canvas 2D context, you need to account for the current transformation matrix.

Core Concept
The mouse event gives coordinates relative to the canvas element, but these do not account for transformations you have applied (scale, translate, rotate, etc.).

The canvas 2D context tracks these transforms internally.

You can use context.getTransform() to get the current transformation matrix.

To get the actual position in your transformed canvas coordinate system, you apply the inverse of this transformation to the mouse position.

How to Get Transformed Mouse Coordinates
js
```
function getTransformedPoint(canvas, event) {
  const ctx = canvas.getContext('2d');
  // Mouse position relative to canvas element, accounting for CSS size scaling
  const rect = canvas.getBoundingClientRect();
  const x = (event.clientX - rect.left) * (canvas.width / rect.width);
  const y = (event.clientY - rect.top) * (canvas.height / rect.height);

  // Get the current transform matrix
  const transform = ctx.getTransform();
  // Invert the matrix
  const inverse = transform.invertSelf();

  // Apply inverse transform to mouse coordinates
  const transformedX = x * inverse.a + y * inverse.c + inverse.e;
```

```
  const transformedY = x * inverse.b + y * inverse.d + inverse.f;

  return { x: transformedX, y: transformedY };
}
```

Explanation

getBoundingClientRect() gets the element's position and size in the page.

Multiply mouse event position by the ratio canvas.width / rect.width to handle different CSS sizes.

getTransform() returns DOMMatrix with properties a, b, c, d, e, f representing the current affine transform.

invertSelf() gets the inverse transformation matrix.

Multiply the mouse point by this inverse matrix to map from screen space back to the transformed canvas space.

Usage Example

js

```
canvas.addEventListener('mousemove', event => {
  const pos = getTransformedPoint(canvas, event);
  console.log('Transformed canvas coordinates:', pos);
});
```

References

Detailed explanation and example:
https://roblouie.com/article/617/transforming-mouse-coordinates-to-canvas-coordinates/

Stack Overflow invert transform example:
https://stackoverflow.com/questions/59245199/get-canvas-mouse-coordinates-after-transformation-using-ctx-gettransformation

MDN CanvasRenderingContext2D.getTransform() docs

This method ensures your input coordinates match the transformed content on canvas exactly, important for panning, zooming, drawing apps, and interactive visualizations.