Transformações Geométricas 2D – Parte 3

Pedro Ximenes

Universidade Católica de Pernambuco (UNICAP)

12 de março de 2026

Sumário

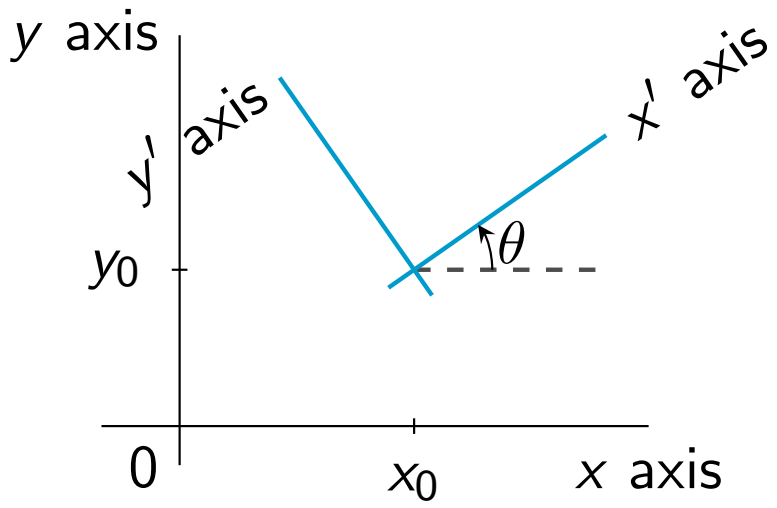
Por que Transformar entre Sistemas de Coordenadas?

Contexto

Em Computação Gráfica, muitas vezes precisamos converter as coordenadas de um ponto de um sistema de coordenadas para outro.

- Mais adiante no curso, vamos aprender **Transformações entre Sistemas de Coordenadas 3D**, essencial para o assunto de **Viewing 3D**
- Antes disso, é mais fácil aprender a versão 2D!

Enunciado do Problema

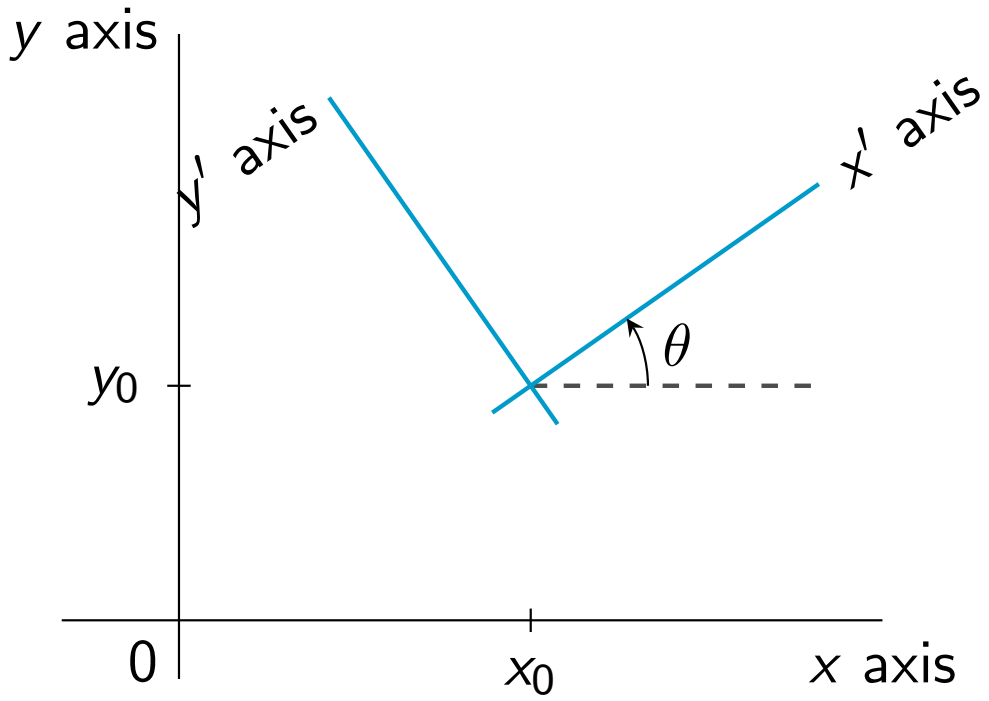


Imagine dois sistemas de coordenadas:

- Sistema **original** XY (em preto)
- Sistema **destino** $X'Y'$ (em azul), cuja origem está em (x_0, y_0) do sistema XY e rotacionado por um ângulo θ

Pergunta: Dado um ponto com coordenadas (x, y) no sistema XY, quais são suas coordenadas no sistema $X'Y'$?

Visualização do Problema



Objetivo

Encontrar as coordenadas de P no sistema $X'Y'$, ou seja, $P' = (x', y')$.
Os dois sistemas representam **a mesma cena**, com os mesmos objetos. Estamos apenas "reindexando" as coordenadas!

Método 1 – Usando o Ângulo θ

Para transformar de XY para $X'Y'$, usamos **duas operações**:

1. **Transladar** a origem de $X'Y'$ (que está em (x_0, y_0)) para coincidir com a origem de XY: $T(-x_0, -y_0)$
2. **Rotacionar** por $-\theta$ para alinhar os eixos: $R(-\theta)$

Matriz Composta

$$M_{XY \to X'Y'} = \underbrace{R(-\theta)}_{2^{\text{a}}\text{ op.}} \cdot \underbrace{T(-x_0, -y_0)}_{1^{\text{a}}\text{ op.}}$$

Lembre: de trás para frente! A translação é a 1ª operação e aparece por **último** na equação.

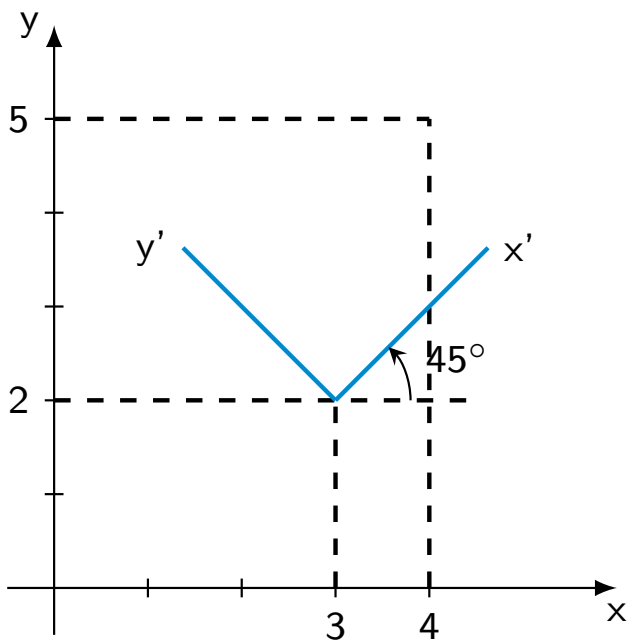
Aplicando ao ponto

$$P' = M_{XY \to X'Y'} \cdot P = R(-\theta) \cdot T(-x_0, -y_0) \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

O resultado P' contém as coordenadas do ponto agora no sistema $X'Y'$.

Exemplo – Método 1

Calcular a matriz de transformação de XY para $X'Y'$ e as coordenadas finais do ponto P no sistema destino (P'). **Dados:** $x_0 = 3$, $y_0 = 2$, $\theta = 45°$, $P = (4, 5)$. Sabendo que $\cos 45° = \sin 45° = \frac{\sqrt{2}}{2}$.



Passo 1: Montar as matrizes (de trás para frente):

$$M = R(-45^\circ) \cdot T(-3, -2) = \begin{pmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & -3 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{pmatrix}$$

Passo 2: Multiplicar as matrizes:

$$M = \begin{pmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & \frac{-5\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ 0 & 0 & 1 \end{pmatrix}$$

Passo 3: Aplicar ao ponto $P = (4, 5, 1)^T$:

$$P' = M \cdot \begin{pmatrix} 4 \\ 5 \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{4\sqrt{2}}{2} \\ \frac{2\sqrt{2}}{2} \\ 1 \end{pmatrix} \approx \begin{pmatrix} 2{,}8284 \\ 1{,}4142 \\ 1 \end{pmatrix}$$

Exemplo – Método 1 (Notas sobre a Rotação $R(-\theta)$)

Por que $-\theta$ e não $+\theta$?

Os dois sistemas estão **desalinhados** por um ângulo θ no sentido anti-horário. Para alinhar, precisamos rotacionar no sentido **horário**, o que significa ângulo **negativo**: $-\theta$.

Lembrete: $R(-\theta)$

$$R(-\theta) = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Como $\cos(-\theta) = \cos\theta$ e $\sin(-\theta) = -\sin\theta$, o sinal do seno "troca de posição" na matriz.

Problema deste método

Geralmente, o ângulo θ **não é informado diretamente**! Nas aplicações futuras, teremos (x_0, y_0) e um segundo ponto, mas não o ângulo.
$\Rightarrow$ Método 2 resolve isso!

Propriedade Ortonormal da Submatriz 2×2

Observação importante

Em qualquer matriz de rotação 3×3 (coordenadas homogêneas), a submatriz 2×2 superior esquerda forma um par de vetores **ortonormais**.

Considere a submatriz de rotação:

$$\begin{pmatrix} r_{xx} & r_{xy} \\ r_{yx} & r_{yy} \end{pmatrix} \quad \Rightarrow \quad \vec{e}_1 = \begin{pmatrix} r_{xx} \\ r_{xy} \end{pmatrix}, \quad \vec{e}_2 = \begin{pmatrix} r_{yx} \\ r_{yy} \end{pmatrix}$$

O que significa "ortonormal"?

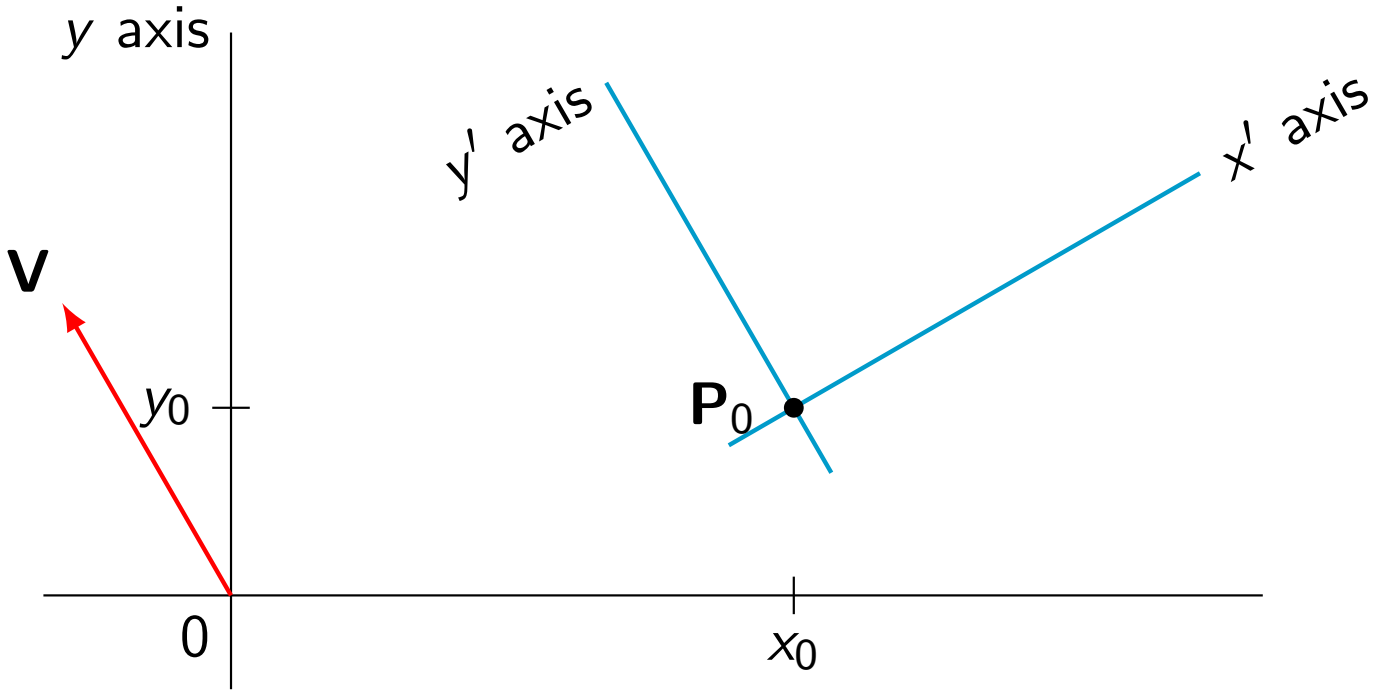
1. **Unitários:** $\|\vec{e}_1\| = 1$ e $\|\vec{e}_2\| = 1$ $\quad$ (norma $= \sqrt{r_{xx}^2 + r_{xy}^2} = 1$)
2. **Ortogonais:** $\vec{e}_1 \cdot \vec{e}_2 = 0$ $\quad$ (produto escalar $= r_{xx} \cdot r_{yx} + r_{xy} \cdot r_{yy} = 0$)

Ou seja, os dois vetores são perpendiculares (90°) e de comprimento 1.

Para que serve isso?

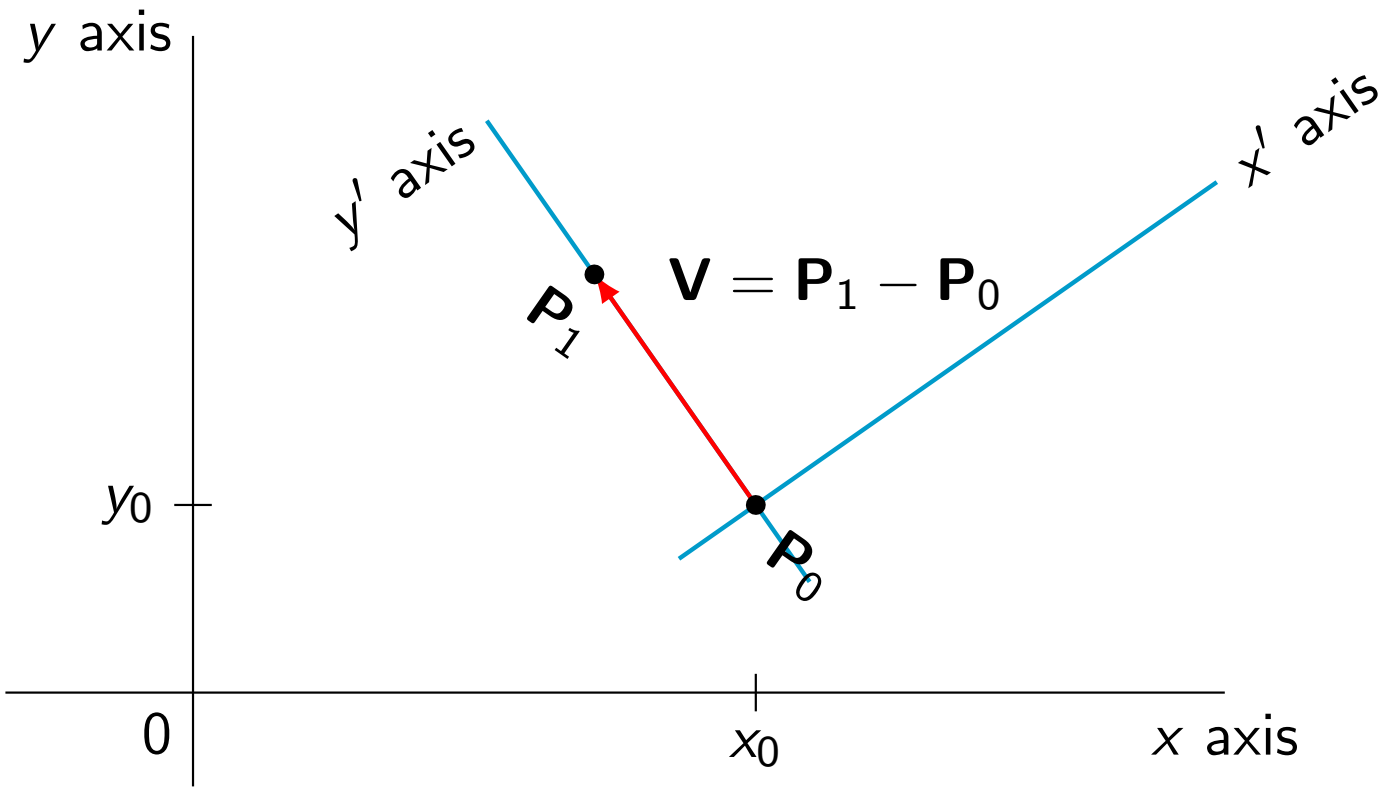
Essa propriedade permite construir a matriz de rotação **sem conhecer** o ângulo θ – é a base do Método 2!

- Usando essa propriedade, outro método para transformar xy em $x'y'$ pode ser derivado:
  - Para isso, inicialmente descrevemos a orientação do sistema de coordenadas $x'y'$ por meio de um vetor V indicando a direção positiva do eixo y'. **i.e um vetor que aponte na mesma direção de y' e tenha a mesma orientação.**



- É possível especificar v com base em P_0 e P_1:

$$v = P_1 - P_0 = (V_x, V_y)$$



Método 2 – Sem Precisar do Ângulo θ

Ideia

Ao invés do ângulo θ, usamos **dois pontos** para definir a orientação do sistema $X'Y'$: a origem $P_0 = (x_0, y_0)$ e um ponto $P_1 = (x_1, y_1)$ na direção positiva do eixo Y'.

Passo 1: Calcular o vetor direção de Y':

$$\vec{V} = P_1 - P_0 = (X_1 - x_0,\ Y_1 - y_0) = (V_X, V_Y) \quad \text{(não normalizado!)}$$

Passo 2: Normalizar o vetor (dividir pela norma):

$$\|\vec{V}\| = \sqrt{V_X^2 + V_Y^2}, \qquad v_x = \frac{V_X}{\|\vec{V}\|}, \quad v_y = \frac{V_Y}{\|\vec{V}\|}$$

Vetores unitários sempre!

Em Computação Gráfica, **sempre normalize** seus vetores! Se um vetor não tem norma 1, normalize-o antes de usar.
Letras maiúsculas (V_X, V_Y) = não normalizado. Minúsculas (v_x, v_y) = normalizado.

Método 2 – Construindo a Matriz de Rotação

Com o vetor normalizado $\vec{v} = (v_x, v_y)$ representando Y', precisamos de um segundo vetor ortogonal para X'.

Construindo o vetor ortogonal

Um vetor perpendicular a (v_x, v_y) é simplesmente: $\vec{u} = (v_y, -v_x)$.
Verificação: $\vec{u} \cdot \vec{v} = v_y \cdot v_x + (-v_x) \cdot v_y = 0$ ✓ (ortogonais!)

A **matriz de rotação** (sem depender de θ) fica: $R = \begin{pmatrix} u_y & u_y & 0 \\ v_x & v_y & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} v_y & -v_x & 0 \\ v_x & v_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$

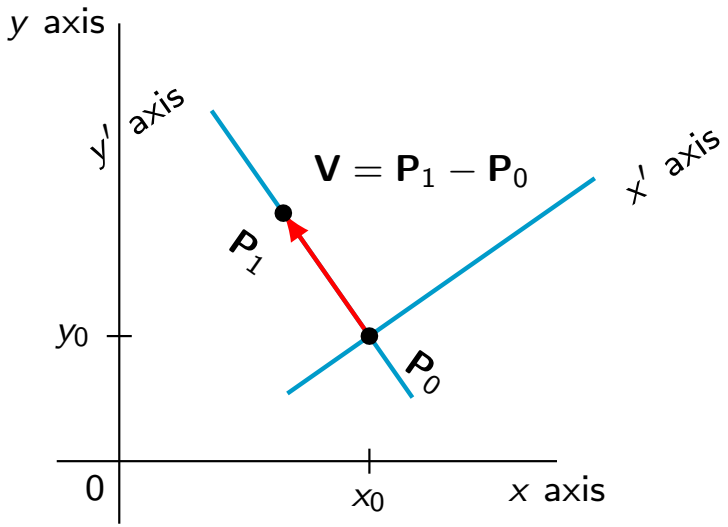
Matriz composta final

A translação **ainda é necessária**! Só substituímos a rotação:

$$M_{XY \to X'Y'} = R \cdot T(-x_0, -y_0) = \begin{pmatrix} v_y & -v_x & 0 \\ v_x & v_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & -x_0 \\ 0 & 1 & -y_0 \\ 0 & 0 & 1 \end{pmatrix}$$

Exemplo – Método 2

Calcular a matriz de transformação de XY para $X'Y'$ e as coordenadas finais do ponto P no sistema destino (P'). **Dados:** $P_0 = (x_0, y_0) = (4, 3)$, $P_1 = (x_1, y_1) = (3, 4)$, $P = (-1, 5)$.



Passo 1: Vetor $\vec{V} = P_1 - P_0$:

$$\vec{V} = (3 - 4,\ 4 - 3) = (-1, 1)$$

Passo 2: Norma e normalização:

$$\|\vec{V}\| = \sqrt{(-1)^2 + 1^2} = \sqrt{2}, \quad v_x = \frac{-1}{\sqrt{2}}, \quad v_y = \frac{1}{\sqrt{2}}$$

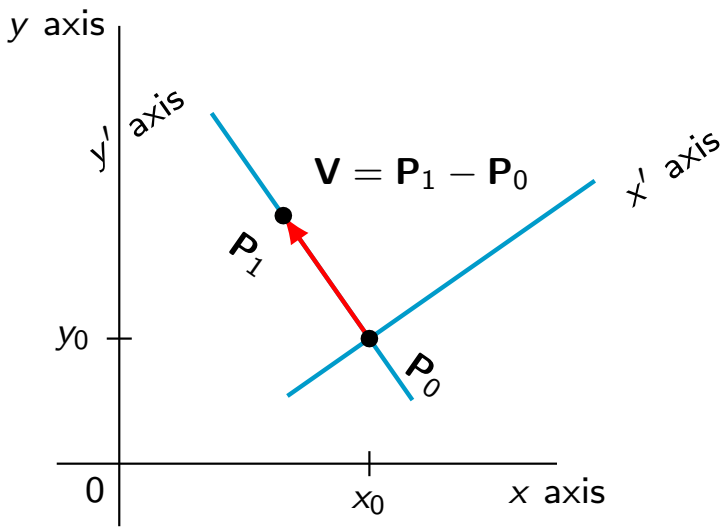
Passo 3: Montar a matriz de rotação:

$$R = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Cuidado com os sinais!

$v_x = -\frac{1}{\sqrt{2}}$, logo $-v_x = -(-\frac{1}{\sqrt{2}}) = +\frac{1}{\sqrt{2}}$.
Dois sinais negativos se cancelam!

Exemplo – Método 2



Passo 4: Translação $T(-4,-3)$ + multiplicação:

$$M = R \cdot T(-4,-3) = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & -4 \\ 0 & 1 & -3 \\ 0 & 0 & 1 \end{pmatrix}$$

Resultado da multiplicação:

$$M = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{-7}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & 0 & 1 \end{pmatrix}$$

Passo 5: Aplicar ao ponto $P = (-1,5,1)^T$:

$$P' = M \cdot \begin{pmatrix} -1 \\ 5 \\ 1 \end{pmatrix} == \begin{pmatrix} \frac{-3}{\sqrt{2}} \\ \frac{7}{\sqrt{2}} \\ 1 \end{pmatrix} \approx \begin{pmatrix} -2{,}12 \\ 4{,}95 \\ 1 \end{pmatrix}$$

Exemplo – Método 2 (Resultado)

Comparação dos Dois Métodos

Método 1 (com θ)

- Precisa saber o ângulo θ
- Usa $R(-\theta)$ diretamente
- Mais simples de entender
- Geralmente usado em exercícios teóricos

Método 2 (sem θ)

- Só precisa de dois pontos: P_0 e P_1
- Constrói vetores ortonormais
- Mais prático em aplicações reais
- Usado quando θ não é dado

Quando usar cada método?

- Se o ângulo θ é dado $\Rightarrow$ **Método 1**
- Se dois pontos do eixo são dados (sem ângulo) $\Rightarrow$ **Método 2**

Ambos os métodos produzem o **mesmo resultado**! A diferença é apenas **como** se obtém a matriz de rotação.

Transformações no Canvas 2D – Introdução

Canvas 2D do HTML5

O elemento `<canvas>` do HTML5 oferece um contexto 2D (`ctx`) com funções nativas de transformação que aplicam matrizes 3×3 internamente.

Três funções principais

1. `ctx.translate(tx, ty)` – Translação
2. `ctx.rotate(angulo)` – Rotação (ângulo em **radianos**!)
3. `ctx.scale(sx, sy)` – Escala

Atenção: ângulo em radianos!

A função `ctx.rotate()` recebe o ângulo em **radianos**, não em graus!
Conversão: $\text{rad} = \text{graus} \times \pi/180$
Exemplo: $90^\circ = 90 \times \pi/180 = \pi/2 \approx 1{,}5708$ rad

ctx.translate – Translação

Sintaxe

`ctx.translate(tx, ty);`
Translada o sistema de coordenadas por (t_x, t_y).

Exemplo

```
// Move o sistema de coordenadas 100px para direita e 50px para baixo
ctx.translate(100, 50);
// Tudo desenhado a seguir estara deslocado
ctx.fillRect(0, 0, 40, 30); // aparece em (100, 50)
```

Observação importante

No Canvas 2D, o eixo y cresce **para baixo** (diferente da convenção matemática). Isso afeta o sentido das rotações e translações verticais!

ctx.rotate e ctx.scale

ctx.rotate(angulo)

Rotação ao redor da **origem** do sistema atual. Ângulo em **radianos**.

```
ctx.rotate(Math.PI / 4);    // rotação de 45°
```

ctx.scale(sx, sy)

Escala a partir da **origem**. Pode mover o objeto (efeito colateral).

```
ctx.scale(2, 0.5);    // duplica em x, reduz à metade em y
```

Cuidados com ctx.scale

- Essa é a escala **sem ponto fixo** (pode mover o objeto!)
- Para escala com ponto fixo: composição (translate $\rightarrow$ scale $\rightarrow$ translate)
- Valores negativos produzem **reflexão**

Cisalhamento

O Canvas 2D não tem função de cisalhamento direta. Use `ctx.transform(a,b,c,d,e,f)` para passar uma matriz 3×3 customizada.

ctx.setTransform e ctx.save/restore

Reset da matriz de transformação

`ctx.setTransform(1, 0, 0, 1, 0, 0);` reseta a matriz para a **identidade**.

Salvando e restaurando estado

```
ctx.save();        // salva o estado atual (matriz, cor, etc.)
ctx.translate(100, 100);
ctx.rotate(Math.PI / 4);
// ... desenha algo transformado ...
ctx.restore();     // restaura o estado anterior
```

Resumo das equivalências

Conceito	**Canvas 2D**
Translação	`ctx.translate(tx, ty)`
Rotação	`ctx.rotate(rad)`
Escala	`ctx.scale(sx, sy)`
Carregar identidade	`ctx.setTransform(1,0,0,1,0,0)`
Salvar/Restaurar	`ctx.save()` / `ctx.restore()`

A Importância do Reset da Matriz

O problema da acumulação

Se você está redesenhando repetidamente (ex: animações com `requestAnimationFrame`), as transformações se **acumulam** a cada quadro!

Sem reset

```
function draw() {
  ctx.clearRect(0,0,w,h);
  // SEM reset!
  ctx.scale(2, 2);
  desenhaCasa();
  requestAnimationFrame(draw);
}
```

Escala acumula: $2\times$, $4\times$, $8\times\ldots$

Com reset

```
function draw() {
  ctx.setTransform(1,0,0,1,0,0);
  ctx.clearRect(0,0,w,h);
  ctx.scale(2, 2);
  desenhaCasa();
  requestAnimationFrame(draw);
}
```

Escala aplicada uma única vez.

Regra de ouro

Sempre resete a matriz (`setTransform` ou `save/restore`) antes de aplicar transformações em cada quadro de animação!

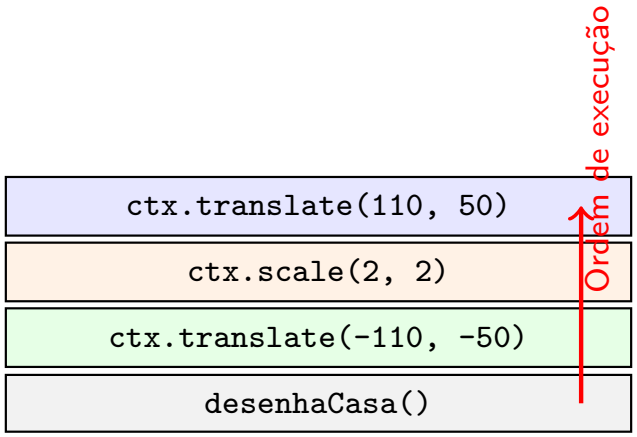
Ordem das Transformações – De Baixo para Cima

Leitura: de baixo para cima!

No Canvas 2D, assim como na teoria, a **primeira transformação aplicada** é a **última linha** de código antes do desenho. As matrizes são **pós-multiplicadas**.

Exemplo: Escala com Ponto Fixo em $(110, 50)$

```
ctx.setTransform(1, 0, 0, 1, 0, 0);
ctx.translate(110, 50);      // 3a operacao (volta)
ctx.scale(2, 2);             // 2a operacao (escala)
ctx.translate(-110, -50);    // 1a operacao (vai p/ origem)
desenhaCasa();                // objeto
```



Exemplo: Rotação com Ponto Arbitrário + Translação

Rotação de 90° ao redor de $(110, 50)$ + Translação $t_x = 50$

```
ctx.setTransform(1, 0, 0, 1, 0, 0);
ctx.translate(50, 0);                  // 4a op: translacao em x
ctx.translate(110, 50);                // 3a op: volta
ctx.rotate(Math.PI / 2);               // 2a op: rotacao 90 graus
ctx.translate(-110, -50);              // 1a op: leva p/ origem
desenhaCasa();
```

A ordem importa!

Se mover `ctx.translate(50, 0)` para **depois** dos três comandos de rotação (antes de `desenhaCasa`), o resultado será **completamente diferente!**
Trocar a posição de **uma única linha** muda o resultado porque a multiplicação de matrizes **não é comutativa**.

Exemplo Completo – Escala com Ponto Fixo

Escala com ponto fixo $(110, 50)$, fator 2

```
1  const canvas = document.getElementById('myCanvas');
2  const ctx = canvas.getContext('2d');
3
4  function desenhaCasa() {
5    ctx.beginPath();
6    ctx.moveTo(100, 100); ctx.lineTo(120, 100);
7    ctx.lineTo(120, 50);  ctx.lineTo(110, 30); // telhado
8    ctx.lineTo(100, 50);  ctx.closePath();
9    ctx.fillStyle = 'red'; ctx.fill();
10 }
11
12 ctx.clearRect(0, 0, canvas.width, canvas.height);
13 ctx.setTransform(1, 0, 0, 1, 0, 0); // reset
14
15 // Escala com ponto fixo (110, 50) - de BAIXO para CIMA
16 ctx.translate(110, 50);     // 3a op: volta
17 ctx.scale(2, 2);           // 2a op: escala
18 ctx.translate(-110, -50);   // 1a op: leva p/ origem
19 desenhaCasa();
```

Composição com Matrizes no WebGL

Do Canvas 2D ao WebGL

Em aplicações WebGL, combinamos as transformações em uma **única matriz** passada ao *vertex shader*.

Composição de matrizes no WebGL

```
var modelView = translate(obj.pos[0], obj.pos[1], 0);
modelView = mult(modelView, rotateZ(obj.theta));
modelView = mult(modelView, scale(obj.sx, obj.sy, 1));
var uMatrix = mult(projection, modelView);
```

A ordem da composição segue a mesma lógica:

$$gl\_Position = \underbrace{projection}_{4^a} \cdot \underbrace{T}_{3^a} \cdot \underbrace{R}_{2^a} \cdot \underbrace{S}_{1^a} \cdot Position$$

Primeira transformação aplicada = última na equação

Escala primeiro, depois rotação, depois translação, por fim projeção.

Transformações 3D com Three.js

Do Canvas 2D ao 3D

Em aplicações 3D, usamos a biblioteca **Three.js**, que abstrai o WebGL e oferece uma API de alto nível. Cada objeto (`Mesh`) possui propriedades de transformação.

Transformações em um objeto Three.js

```
// Escala
mesh.scale.set(2, 2, 1);
// Rotacao (em radianos, eixo Z)
mesh.rotation.z = Math.PI / 4;
// Translacao
mesh.position.set(110, 50, 0);
```

O Three.js compõe automaticamente a matriz `mesh.matrix` na ordem:

$$M = \underbrace{T}_{\text{position}} \cdot \underbrace{R}_{\text{rotation}} \cdot \underbrace{S}_{\text{scale}}$$

Mesma lógica de composição

Escala primeiro, depois rotação, depois translação — a mesma ordem que vimos no Canvas 2D.

Referências


Angel, E.; Shreiner, D. *Interactive Computer Graphics: A Top-Down Approach with WebGL*. 8th edition. Capítulo 5.


Mount, D. *CMSC 427 – Computer Graphics Lecture Notes*. University of Maryland. Aulas 6–9.


Material de Introdução à Computação Gráfica – IME-USP. Disponível em: `https://panda.ime.usp.br/introcg/static/introcg`