Em um jogo é comum que existam objetos que são derivados de um mesmo modelo, como por exemplo árvores em uma floresta. Para buscar otimização de memória são utilizadas instâncias de um mesmo modelo. Instâncias são alterações em modelos baseados em parâmetros pré-definidos. Dessa maneira, ao invés de guardar várias cópias de um mesmo modelo, basta guardar as alterações sobre esse modelo.



Pelo que foi visto acima existem duas principais vantagens na utilização de instâncias. São elas:

* Para representar objetos diferentes basta armazenar o nome do modelo e os parâmetros de instanciamento, sem precisar repetir o modelo;
* Permitir um grande número de objetos diferentes a partir de um mesmo modelo.

Isso cria a necessidade de transformar essas instâncias de objetos em diversos sistemas de coordenadas já que é necessário criar a ilusão de que a cena é construída com diversos objetos.

# **Sistemas de Coordenadas**

Uma instância de objeto é transformada em diversos sistemas de coordenadas até que seja renderizada na tela do computador. Cada um dos sistemas de coordenadas tem uma função bem específica.

[Na Unity em shaders...](https://docs.unity3d.com/Manual/SL-BuiltinFunctions.html)

Antigamente existiam essas funções mais "visíveis" para os programadores de plantão para manipulação de vértices. Nas versões novas os shaders foram atualizados

## **Sistemas de Coordenadas do Objeto**

O sistema de coordenadas do objeto é simplesmente o sistema de coordenadas em que o modelo foi criado. Basicamente, se utiliza de um sistema de coordenadas que facilite a criação do modelo em algum software de modelagem 3D.

## **Sistemas de Coordenadas Hierárquico**

Em algumas situações é comum que a cena seja construída de forma hierárquica, ou seja, a posição de um objeto é relativa ao parente, ao invés de ser relativa às coordenadas do mundo. Por exemplo, a mão pode ser posicionada relativa ao braço e o braço relativo ao torso. Quando o braço se movimenta, a mão se movimenta junto e quanto o torço se movimenta os três objetos são movidos juntos.

## **Sistemas de Coordenadas do Mundo**

O sistema de coordenadas do mundo é um sistema universal que permite que cada modelo interaja com o outro. Para que a interação ocorra, o sistema de coordenadas de cada modelo em cena é transformado no sistema de coordenadas do mundo.

## **Sistemas de Coordenadas da Câmera**

O que é visto na tela é relativo ao observador, mais especificamente à câmera. A mudança na orientação e posicionamento da câmera modifica diretamente o que o observador está vendo. O sistema de coordenadas do mundo é transformado no sistema de coordenadas da câmera e esse sistema de coordenadas define o que é mostrado na tela.

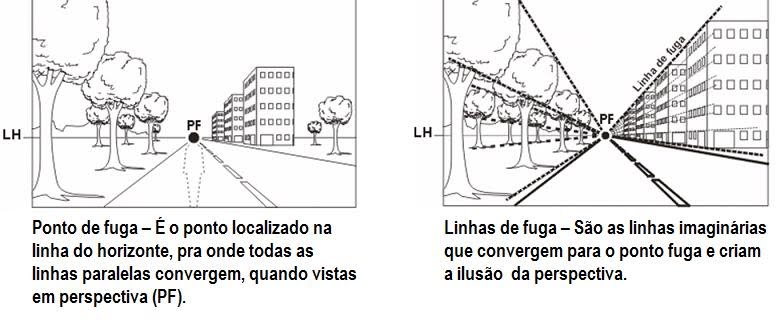
## **Sistemas de Coordenadas de Projeção**

O que é mostrado na tela como tridimensional na realidade é uma ilusão, já que o que realmente está sendo mostrado é uma imagem bidimensional. A transformação final dentro das APIs gráficas é a projeção dos objetos. Essa transformação converte um cenário tridimensional em uma imagem bidimensional. O sistema de coordenadas de projeção pode ser configurado de duas formas: A Visão Perspectiva ou Visão Orthogonal.

### **Visão Perspectiva (Perspective View):**

Quando um sistema de coordenadas é configurado como perspectiva é criada uma ilusão de tridimensionalidade. Essa ilusão é criada utilizando pontos de fuga (como os de desenho arquitetônico!) e fazendo os objetos que estão mais afastados da câmera menores, criando assim uma ilusão de profundidade.

Essa câmera é comumente utilizada para jogos 3D.



### **Visão Orthogonal (Orthogonal View):**

Na visão Ortogonal, cada objeto da cena é visto como bi-dimensional e não é criada uma ilusão de profundidade. É comum a utilização dessa câmera em jogos 2D e na UI de jogos 3D.

# **Transformações Geométricas**

## **Translação**

A translação determina a posição do modelo no universo. Dadas as coordenadas (x0, y0) de um vértice do modelo no sistema de coordenadas do objeto (SCO) e os parâmetros da translação desse modelo no eixo X e Y (TX, TY), a obtenção do novo par de coordenadas (xu, yu) no sistema de coordenadas do mundo (SCM) é:

***xu = x0 + TX***

***yu = y0 + TY***

Destaca-se que qualquer vértice que esteja na origem no SCO será transladado para o ponto (TX,TY) no SCM.

<Desenhar um Triângulo ABC qualquer no plano e executar a translação>

Em OpenGL é usado o comando glTranslatef( dx, dy, dz) ou o comando glTranslated( dx, dy, dz). Observe que no OpenGL a translação irá afetar qualquer desenho que seja feito após o comando ser dado. A primeira versão do comando espera receber 3 valores float, enquanto a segunda espera 3 double.

## **Escala**

A escala determina a alteração no tamanho do modelo e a fórmula utilizada para "escalar" um vértice é:

***xu = x0 \* EX***

***yu = y0 \* EY***

A escala sempre ocorre em torno do SCO e, portanto, qualquer vértice que estiver na origem no SCO continuará nessa posição no SCM. Entretanto, quaisquer outros vértices do modelo que não estejam na origem sofrem um deslocamento em relação a esta após a operação de escala. Esse ponto deve ser considerado em relação a esta após a operação de escala. Esse ponto deve ser considerado durante a construção do modelo já que é possível que translações indesejáveis ocorram.

<Desenhar o mesmo Triângulo ABC e executar a escala sozinha, translação e escala, escala e translação>

Em OpenGL a escala funciona de forma similar a translação, entretanto, ao invés de ser passado apenas 1 fator de escala são passados 3 fatores. O comando na OpenGL para escalar um objeto é o glScalef(sx, sy, sz) ou ainda glScaled(sx, sy, sz). Esses comandos transformam um ponto (x, y, z) em (x\*sx , y \*sy, z \*sz). Observe que é importante que a escala seja ajustada antes do objeto ser transladado!

#### **Escala em torno de um ponto arbitrário:**

Em algumas situações é necessário fazer alteração do tamanho de uma instância de um modelo em relação a um ponto diferente da origem no SCO. Chama-se este ponto de Ponto de Referência (PR). Tomando PR, com coordenadas (x,y), como ponto em relação que deseja escalar um objeto e PO um vértice qualquer do modelo com coordenadas (x0, y0):

1. Aplica-se sobre PO a translação que seria necessária caso estivéssemos transladando PR para a origem:

***xa = x0 - xr***

***ya = y0 - yr***

2. Aplica-se sobre o ponto obtido a escala desejada

***xb = xa \* EX***

***yb = ya \* EY***

3. Desfazer a translação realizada no passo 1:

***xu = xb + xr***

***yu = yb + yr***

Se unirmos os três passos em um única fórmula obtemos:

***xu = (x0 - xr) \* EX + xr***

***yu = (y0 - yr)\* EY + yr***

## **Rotação**

A rotação determina a orientação de uma instância dentro do SCM. A fórmula de rotação de um ponto (x0, y0) em um ângulo beta é dada por:

**xu = x0 \* cos(beta) - y0 \* sin(beta)**

**yu = y0 \* cos(beta) + x0 \* sin(beta)**

Da mesma forma que a escala, a rotação também trabalha em torno da origem.

< Desenhar o mesmo triângulo e executar a rotação sozinha, translação e rotação e rotação e translação >

Na OpenGL o comando glRotatef(r, ax, ay, az) rotaciona um objeto onde o primeiro parâmetro determina o ângulo de rotação, medido em graus, enquanto os outros 3 parâmetros determinam o eixo de rotação.

#### **Rotação em torno de um ponto arbitrário:**

Assim como a escala, existem situações em que é necessário rotacionar um modelo em relação a um ponto diferente da origem no SCM. Tomando PR, com coordenadas (x,y), como ponto em relação que se deseja rotacionar um objeto e PO um vértice qualquer do modelo de coordenadas (x0, y0):

1. Aplicar sobre PO a translação que seria necessária caso estivéssemos transladando PR para a origem.

***xa = x0 - xr***

***ya = y0 - yr***

2. Aplicar sobre o ponto obtido a rotação desejada

***xb = xa \* cos(beta) - ya \* sin(beta)***

***yb = ya \* cos(beta) + xa \* sin(beta)***

3. Desfazer a translação realizada no passo 1:

***xu = xb + xr***

***yu = yb + yr***

Se unirmos os passos em uma fórmula temos:

***xu = (x0 - xr) \* cos(beta) - (y0 - yr) \* sin(beta)***

***yu = (y0 - yr) \* cos(beta) + (x0 - xr) \* sin(beta)***

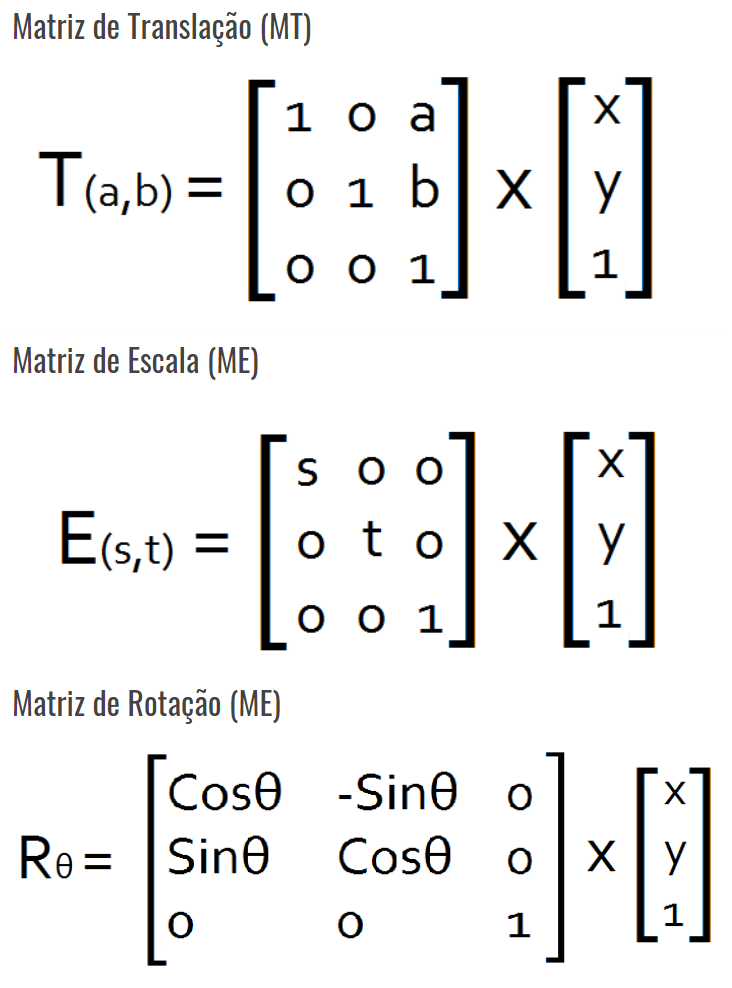
## **Matrizes de Transformação Geométrica**

Se uma matriz A e B são 2 matrizes N x N, então elas podem ser multiplicadas e isso resulta em C = AB, onde C também é uma matriz. Se um vetor v tiver comprimento N, então v pode ser multiplicado por A e produz 1 novo vetor w, onde w = Av. Agora considere também a matriz B que também tem tamanho N x N. A partir disso é possível formar dois produtos diferentes: A (Bv) e (AB)v. É importante destacar que o resultado dessas duas operações é o mesmo. Ou seja, se multiplicarmos v por B e então multiplicarmos por A obteremos o mesmo resultado se multiplicarmos as matrizes A e B, e em seguida, multiplicarmos por v. É importante destacar que uma função que transforma qualquer vetor com comprimento N em outro vetor de comprimento N é chamada de **transformação linear.**

A rotação e a escala são transformações lineares. Ou seja, a operação de rotação de (x,y) por um angulo d, considerando que o objeto está na origem, pode ser feita por uma matriz 2x2 que iremos chamar de Rd. Da mesma forma, a escala por um fator a na horizontal e b na vertical pode ser dada por uma matriz Sab. Se nós quisermos aplicar rotação e escala em um vértice v = (x,y) que pertence a um objeto posicionado na origem na origem é possível aplicar Rd(Sab\*v) ou ainda (Rd\*Sab)\*v.

Portanto, não é obrigatório para o computador manter as duas operações separadas se o objetivo for aplicar essa operação para um objeto complexo.

Entretanto, observe que nada foi dito da translação! **A translação não é uma transformação linear!** Para incluir a translação é feito o seguinte: ao invés de representar um ponto 2D como um par ordenado (x,y), nós representamos como uma tríplice de números (x,y,1). Portanto, é adicionada uma terceira coordenada. Com isso é possível representar a rotação, a escala e a translação, assim como qualquer outra transformação afim, de qualquer vértice no espaço 2D por através da multiplicação de uma matriz 3x3.



< Mostrar a execução para cada 1 das matrizes de transformação >