Vector/Vector3

The **vector3** class represents a three-dimensional vector.

Contents

Methods

create

This is default constructor for the Vector3 class and returns a Vector3 object.

Syntax

```
vector3 Vector3 ( [ float x = 0, float y = 0, float z = 0 ] )
```

Optional arguments

- **x**, **y** and **z**: coordinates for the vector. If not specified, they default to 0.
- Instead of these three coordinates, a single Vector3 object may be inserted to clone it.

Example

This example sorts all players in a nice line on the center of the map.

Shared

```
local players = getElementsByType("player")
local newPlayerPosition = Vector3(-#players - 1, 0, 10) -- Initialize the position vector for the first player in the list
for _, player in ipairs(players) do
    -- Move each player 1 unit forward in X from the previous one
    newPlayerPosition.x = newPlayerPosition.x + 1
    setElementPosition(player, newPlayerPosition)
end
```

cross

Calculates the cross product of two vectors, A and B, and is written as A × B. The result is another vector which:

- Is orthogonal to both A and B.
- Its direction is determined by the right-hand rule.
- Its length is equal to the area of the parallelogram that A and B form (which in turn is equal to A's length by B's length by the sine of the minimum angle between A and B).

Syntax

```
vector3 Vector3:cross ( vector3 vector )
```

Required arguments

- **vector**: a vector3 object to get the cross product with.

Example

This example creates a tiny sun which is always at the right of the vehicle the local player drives. By default it will draw the vectors used to compute the position where it should be, so you can understand what does this function do. Note that the code could be made simpler by simply getting the right component of the vehicle matrix. **Make sure that you put this code in a resource which has OOP enabled, or it won't work**.

Client

```
-- OPTIONS --
local debugMode = true -- If true, the script will draw the vectors used to compute the effect

-- EFFECT CONTROL FUNCTIONS --
local playerVehicle, lightMarker, light
local function applyDrivingLightEffect()
    -- Get the vehicle matrix and use it to get the vehicle and vehicle base position
    local vehicleMatrix = playerVehicle.matrix
    local vehiclePosition = vehicleMatrix:getPosition()
    -- Get the vehicle upwards vector and vehicle forward direction vector
    local vehicleUpwardsVector, vehicleForwardVector = vehicleMatrix:getUp(), vehicleMatrix:getForward()
    -- Get the normalized cross product of the vehicle forward vector and the vehicleUpwardsVector vector
    local crossVector = vehicleForwardVector:cross(vehicleUpwardsVector):getNormalized()
```

```
    -- Draw all the interesting vectors we have now
    if debugMode then
        local forwardPoint = vehiclePosition + vehicleForwardVector
        dxDrawLine3D(vehiclePosition, forwardPoint, tocolor(255, 0, 0))
        local sx, sy = getScreenFromWorldPosition(forwardPoint)
        if sx then
            dxDrawText("F", sx, sy)
        end

        local vehicleUpwardsVectorPoint = vehiclePosition + vehicleUpwardsVector
        dxDrawLine3D(vehiclePosition, vehicleUpwardsVectorPoint, tocolor(0, 255, 0))
        local sx, sy = getScreenFromWorldPosition(vehicleUpwardsVectorPoint, 1.1)
        if sx then
            dxDrawText("U", sx, sy)
        end

        local crossPoint = vehiclePosition + crossVector
        dxDrawLine3D(vehiclePosition, crossPoint, tocolor(0, 0, 255))
        local sx, sy = getScreenFromWorldPosition(crossPoint)
        if sx then
            dxDrawText("C", sx, sy)
        end
    end
    -- Calculate the half length of the vehicle based in its bounding box, and use it to position the light effects at the right
    local _, bmy, _, _, bMy = playerVehicle:getBoundingBox()
    local newEffectPosition = vehiclePosition + crossVector * (bMy - bmy) / 2
    lightMarker.position = newEffectPosition
    light.position = newEffectPosition
end

local function startDrivingLightEffect()
    -- Create the effects and start updating them every frame
    addEventHandler("onClientPreRender", root, applyDrivingLightEffect)
    playerVehicle, lightMarker, light = localPlayer.vehicle, createMarker(0, 0, 0, "corona", 0.5, 255, 255, 0), createLight(0, 0, 0, 0, 8, 255, 255)
    -- If we are in debug mode, render the vehicle invisible to see the vectors clearly
    if debugMode then
        playerVehicle.alpha = 0
    end
end

local function stopDrivingLightEffect()
    -- Stop applyDrivingLightEffect from being called and destroy everything created
    removeEventHandler("onClientPreRender", root, applyDrivingLightEffect)
    destroyElement(lightMarker)
    destroyElement(light)
    -- If we are in debug mode, reset the vehicle alpha to normal again
    if debugMode then
        playerVehicle.alpha = 255
    end
    playerVehicle, lightMarker, light = nil, nil, nil
end

-- FUNCTIONS THAT MANAGE THE EFFECT --
-- Start or stop the effect when the player is driving a car
local function manageDrivingLightEffectStatus(_, seat)
    if eventName == "onClientPlayerVehicleEnter" then
        if seat == 0 then
            -- The player has just entered a vehicle as the driver. Start the effect
            startDrivingLightEffect()
        end
    elseif playerVehicle then
        -- The player has just exited a vehicle and we were applying the effect. Stop it
        stopDrivingLightEffect()
    end
end
addEventHandler("onClientPlayerVehicleEnter", localPlayer, manageDrivingLightEffectStatus)
addEventHandler("onClientPlayerVehicleExit", localPlayer, manageDrivingLightEffectStatus)

-- Start the effect when the resource starts if the player is driving a vehicle, and reset vehicle alpha back to normal if necessary
local function handleResourceStartStop()
    if eventName == "onClientResourceStart" then
        playerVehicle = localPlayer.vehicle or nil
        if playerVehicle then
            startDrivingLightEffect()
        end
    elseif playerVehicle then
        -- It is not necessary to call this function to just reset the car alpha, but it is a good practise to ALWAYS clean up everything nevertheless
        stopDrivingLightEffect()
    end
end
addEventHandler("onClientResourceStart", resourceRoot, handleResourceStartStop)
addEventHandler("onClientResourceStop", resourceRoot, handleResourceStartStop)
```

dot

Calculates the (standard) dot/scalar product of two vectors. If we call that vectors A and B, the dot product is written as A · B. This can be used to calculate the angle between them. If the standard scalar product is 0, both vectors are orthogonal.

Syntax

```
float Vector3:dot ( vector3 vector )
```

Required arguments

- **vector**: a vector3 object to get the dot product with.

Example

This examples illustrates the concept of dot/scalar product and implements a useful function which can be used to get the angle between two vectors.

Shared

```
local vec1 = Vector3(1, 0, 0)
local vec2 = Vector3(0, 0, 0)
local dotproduct = vec1:dot(vec2)

if dotproduct == 0 then
    outputDebugString("vec1 is orthogonal to vec2")
end

-- Calculate angle between vec1 and vec2
```

```
function angle(vec1, vec2)
    -- Calculate the angle by applying law of cosines
    return math.acos(vec1:dot(vec2)/(vec1.length*vec2.length))
end

outputDebugString("Angle between vec1 and vec2: "..math.deg(angle(vec1, vec2))..""")
```

normalize

Converts a vector to a unit vector (a vector of length 1).

Syntax

```
bool Vector3:normalize ( )
```

Example

This example slowly moves all the players' camera to look at the Mount Chilliad.

Client

```
local targetPosition = Vector3(-2627.32, -1083.2, 433.35) -- Somewhere in Mount Chilliad

local function moveCameraToTarget(deltaTime)
    local currentPosition = Vector3(getCameraMatrix())
    local direction = targetPosition - currentPosition
    direction:normalize() -- Get a direction vector by normalizing the vector from the current position to the target position
    setCameraMatrix(currentPosition + direction * deltaTime * 0.05, -2589.45, -1174.49, 418.09)
end
addEventHandler("onClientPreRender", root, moveCameraToTarget)
```

getX and setX

getY and setY

getZ and setZ

getNormalized

Returns a normalized vector (of length 1) of the vector it's used on. Differently from the *Vector3:normalize* method, this one returns a *vector3* and doesn't modify the original vector.

Syntax

```
vector3 Vector3:getNormalized ( )
```

Example

This example slowly moves all the players' camera to look at the Mount Chilliad with a shorter code than the previous example.

Client

```
local targetPosition = Vector3(-2627.32, -1083.2, 433.35) -- Somewhere in Mount Chilliad

local function moveCameraToTarget(deltaTime)
    local currentPosition = Vector3(getCameraMatrix())
    local direction = (targetPosition - currentPosition):getNormalized() -- Get a direction vector by normalizing the vector from the current position to the target position
    setCameraMatrix(currentPosition + direction * deltaTime * 0.05, -2589.45, -1174.49, 418.09)
end
addEventHandler("onClientPreRender", root, moveCameraToTarget)
```

getSquaredLength

getLength

intersectsSegmentTriangle

Returns the intersection of the given triangle, or false if there is no intersection.

Syntax

```
vector3 Vector3.intersectsSegmentTriangle ( vector3 origin, vector3 segmentDir, vector3 triVert0, vector3 triVert1, vector3 triVert2 )

vector3 Vector3:intersectsSegmentTriangle ( vector3 segmentDir, vector3 triVert0, vector3 triVert1, vector3 triVert2 )
```