Client-side function

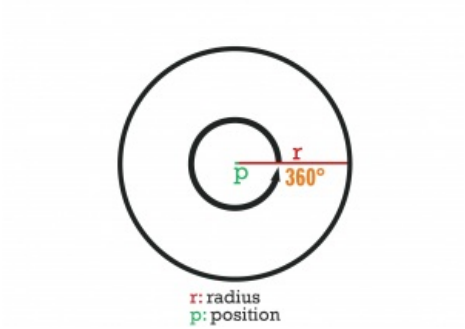
DxDrawCircle

This function draws a circle shape on the screen - rendered for **one** frame. This should be used in conjunction with onClientRender in order to be display continuously.

Syntax

```
bool dxDrawCircle ( float posX, float posY, float radius [, float startAngle = 0.0, float stopAngle = 360.0, int theColor = w
hite,
                    int theCenterColor = theColor, int segments = 32, int ratio = 1, bool postGUI = false ] )
```

Required Arguments



An example of how dxDrawCircle function works in practice.

- **posX**: An integer representing the **absolute** X position of the circle center, represented by pixels on the screen.
- **posY**: An integer representing the **absolute** Y position of the circle center, represented by pixels on the screen.
- **radius**: An integer representing the radius scale of the circle that is being drawn.

Optional Arguments

NOTE: When using optional arguments, you might need to supply all arguments before the one you wish to use. For more information on optional arguments, see optional arguments.

- **startAngle**: An integer representing the angle of the first point of the circle.
- **stopAngle**: An integer representing the angle of the last point of the circle.
- **theColor**: An integer of the hex color, produced using tocolor or 0xAARRGGBB (AA = alpha, RR = red, GG = green, BB = blue).
- **theCenterColor**: An integer of the hex color, produced using tocolor or 0xAARRGGBB (AA = alpha, RR = red, GG = green, BB = blue).
- **segments**: An integer ranging from 3-1024 representing how many triangles are used to form the circle, more segments = smoother circle. Note: using lots of segments may cause lag.
- **ratio**: Ratio between width and height, e.g: 2 would mean that the width of the circle is 2 times the height.
- **postGUI**: A bool representing whether the circle should be drawn on top of or behind any ingame GUI (rendered by CEGUI).

Returns

Returns *true* if the creation of the 2D circle was successful, *false* otherwise.

Remarks

By documentation this function does perform a **triangle approximation of the cicle**. This is important for when there is no shader support on the end-user hardware. If there is shader support then you can instead use shaders with euler-based math (sin/cos/asin) to create a smoother circle. The performance does scale much better than increasing the segment count of this function because drawing using shader would take up only one draw-call. Take look at this post for more details.

The above remark is about drawing circles on modern end-user hardware with angle-interval support, as necessary by example for pie-charts. **Drawing full circles** is easier because the math can be simplified to a euler distance calculation from the texcoord (0.5, 0.5) to the currently drawing pixel. Then the HLSL clip function can be used to not render any pixels outside of the circle. Read more about it in this post.