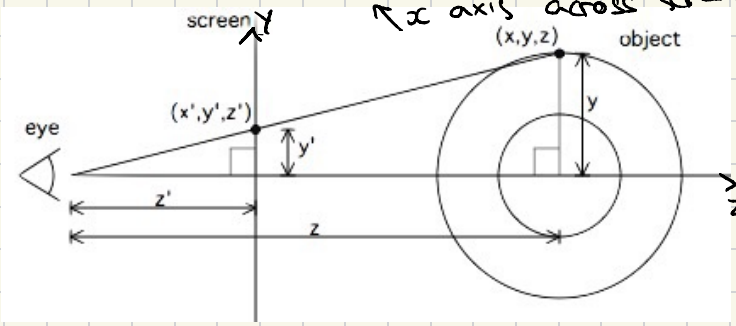
C - Donut

Perspective



To render we project each point into 3-D space. (x, y, z).

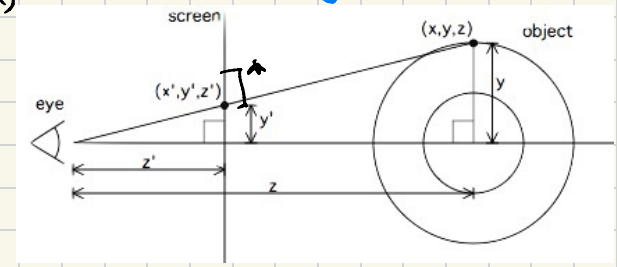
(this is a side view)

- As (x, y, z) & (x', y', z') form right angle triangles relative proportions are maintained.

$$\Rightarrow \frac{y'}{z'} = \frac{y}{z}$$ (sides proportional) ⇒ constant ratio.

$$\Rightarrow y' = \frac{yz'}{z}$$

Screen Projection Understanding



However, z' is a constant for the distance from the screen so it can be d.

$$y' = \frac{dy}{z}$$

* anywhere can be the top of the screen but we are only projecting at y' height to be relative to y. in our space.

∴ Our projection equation becomes

$$(x', y') = \left(\frac{dx}{z}, \frac{dy}{z}\right)$$

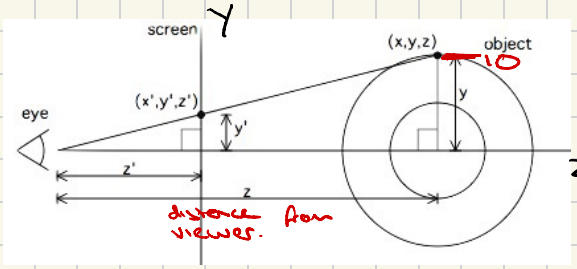
as (x', y') represents the x y coordinate for the pixel on the screen (due to perspective & depth) & math for x coordinate the same.

d

- this represents the field of view we want to show in our 2D window.

An example Projection.

Eg: 100×100 pixels window
⇒ center at (50, 50)
⇒ we want 10 units wide (peak. $x=10, y=10$)
⇒ we want 5 units back from viewer. ⇒ $z = 5$.
∴ we need a d (z') such that our projection is still on the screen.



⇒ Using x-coor : $(x') < 50 = (\frac{dx}{z}) < 50 = \frac{(10)d}{(5)} < 50$

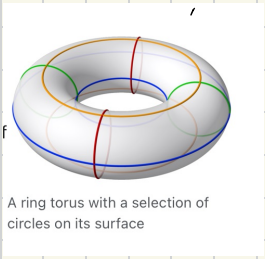
Perspective (Depth)

Some points might be infront of another if plotting on the same pixel.
ie same (x', y') but different z.

⇒ we need a z-buffer.
↳ check whether we are plotting something in front of another.

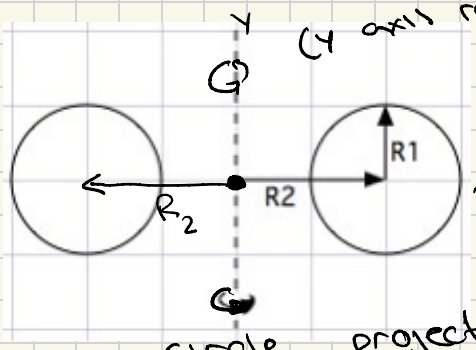
- Using z^{-1} is better as then $z^{-1} = 0$ corresponds to infinite depth.
(it can also be reused for computing $z'(x', y')$)

TORUS


A ring torus with a selection of circles on its surface

- a torus is a solid revolution
  ↳ a solid figure (in this case a circle) rotating a plane figure around some straight line (called axis of revolution.

∴ To draw the torus we need to plot a 2D circle rotating around some point (also a rotational axis) in 3D space



(Y axis rotation)

circle projected around the axis

∴ The cross section is circle centered at R_2 with radius R_1.

$$(x, y, z) = (R_2, 0, 0) + (R_1 \cos\theta, R_1 \sin\theta, 0)$$

for θ is $0 \to 2\pi$ to form the 2D cross section circle.

↳ θ is sweeping angle to form the solid figure in the solid revolution.

Rotation around axis

To rotate an arbitrary 3D point around an axis we use rotational matrices.

Lets call the angle we are rotating about ϕ.

∴ For our solid figure (circle, projected with θ, $0 \to 2\pi$ to form the shape), then that circle rotated around the Y-AXIS by some angle ϕ is:

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

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

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

3D

$$(x, y, z) \cdot R_y(\phi) = [R_2 + R_1\cos\theta, R_1\sin\theta, 0] \begin{bmatrix} \cos\phi & 0 & \sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{bmatrix}$$

Understanding Rotation.

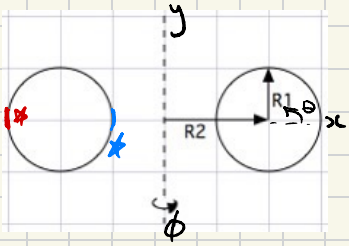
$$R_y(\phi)\,(x,y,z) = \begin{bmatrix} \cos\phi & 0 & \sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{bmatrix} \begin{bmatrix} R_2 + R_1\cos\theta \\ R_1\sin\theta \\ 0 \end{bmatrix}$$

$$\Rightarrow (x,y,z) = \left((R_2 + R_1\cos\theta)\cos\phi,\ R_1\sin\theta,\ -(R_2+R_1\cos\theta)\sin\phi\right)$$

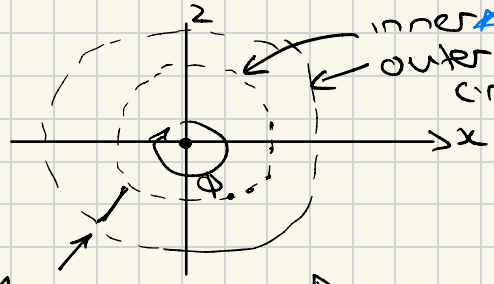
What this is projecting is a torus, rotated around the y-axis. It will not be "rotating" as the image of the shape is constructed around the y-axis, not rotating it (a solid revolution).

$\therefore$ θ varies from $0 \to 2\pi$ to trace the cross sectional circle located on x-y plane

ϕ varies from $0 \to 2\pi$ to trace the cross sectional circle located on x-z plan



x-z cross section perspective.



inner ★ outer part of circle ★

each slice is a circle

each point has been rotated around our y-axis.

$\Rightarrow$ Together, θ, ϕ plot each point on the torus.

Movement Rotation.

To get the torus to rotate we then apply rotational matricies of other axis onto our torus coordinates

$$(R_2 + R_1\cos\theta,\quad R_1\sin\theta,\quad 0)\cdot\begin{pmatrix} \cos\phi & 0 & \sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{pmatrix}\cdot\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos A & \sin A \\ 0 & -\sin A & \cos A \end{pmatrix}\cdot\begin{pmatrix} \cos B & \sin B & 0 \\ -\sin B & \cos B & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

(x,y,z) cross section θ

$R_y(\phi)$ $R_x(A)$ $R_z(B)$

Moving around x & z axis by angle A & B.

Projecting torus on screen.

Overall we now have a torus, rotating around two axis, x & z.

To get screen coordinates:

- we need to move torus somewhere in front of the viewer (some constant d_2)

project from 3D → 2D

$(x, y, z) \rightarrow (x', y')$

$$\therefore \quad (x', y') = \left(\frac{d_1 x}{d_2 + z}, \frac{d_1 y}{d_2 + z} \right)$$

where x, y, z are our coordinates after the rotation of torus

> d_1 & d_2 can be changed to affect field of view and flatten or exaggerate depth

- The computed (x, y, z):

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} (R_2 + R_1 \cos\theta)(\cos B \cos\phi + \sin A \sin B \sin\phi) - R_1 \cos A \sin B \sin\theta \\ (R_2 + R_1 \cos\theta)(\cos\phi \sin B - \cos B \sin A \sin\phi) + R_1 \cos A \cos B \sin\theta \\ \cos A (R_2 + R_1 \cos\theta) \sin\phi + R_1 \sin A \sin\theta \end{pmatrix}$$

Surface Normal & Illumination

The surface normal - direction perpendicular to the surface at each point.

⇒ we can dot product this with the light direction vector (where the light shines from) to produce an angle

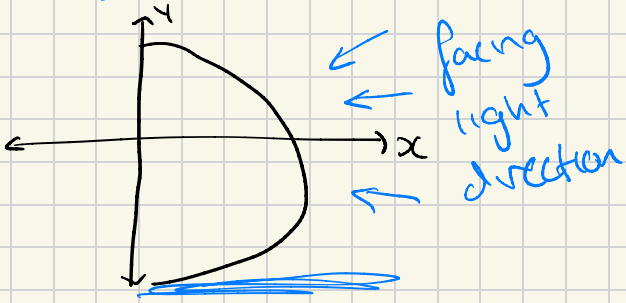
NOTE: Let **a** & **b** be vectors

$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}||\mathbf{b}| \cos\theta$$ → angle between them

∴ If we let the normal surface vector & light vector magnitudes be 1 we get the cosine of the angle between them

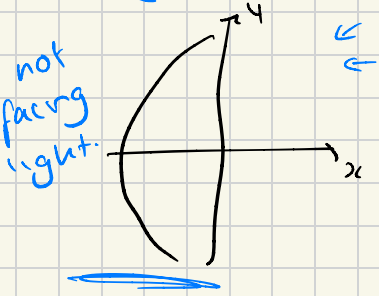
⇒ dot product > 0

$\cos(\theta) > 0$



dot product < 0

$\cos(\theta) < 0$



$$(N_x,\ N_y,\ N_z) = (\cos\theta,\ \sin\theta,\ 0) \cdot \begin{pmatrix} \cos\phi & 0 & \sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos A & \sin A \\ 0 & -\sin A & \cos A \end{pmatrix} \cdot \begin{pmatrix} \cos B & \sin B & 0 \\ -\sin B & \cos B & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

(N_x, N_y, N_z) → surface normal

$(\cos\theta, \sin\theta, 0)$ → radius = 1

→ surface normal is same as torus but with radius 1.

Light Direction Vector $\frac{1}{\sqrt{2}}(0, 1, -1)$

$\frac{1}{\sqrt{2}}$ → normalised to 1.

$(0, 1, -1)$ → above & behind viewer

Luminance (L) is therefore determined by the range $(0 \rightarrow 1)$ that represents the $\cos(\theta)$ of the normal vector & light direction vector:

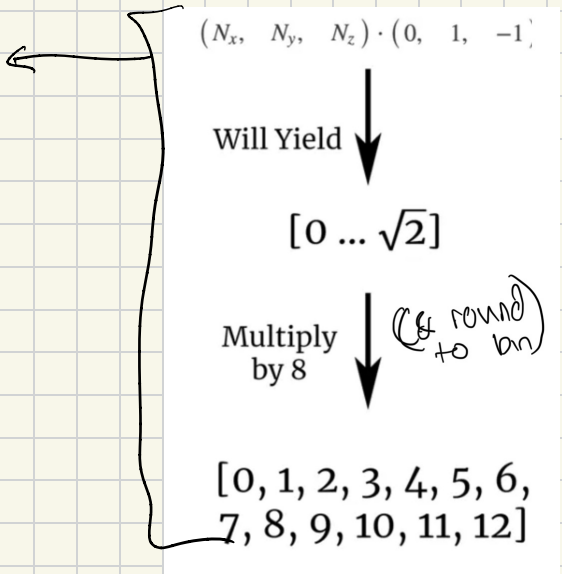
$$L = (N_x, N_y, N_z) \cdot \frac{1}{\sqrt{2}}(0, 1, -1)$$

OR

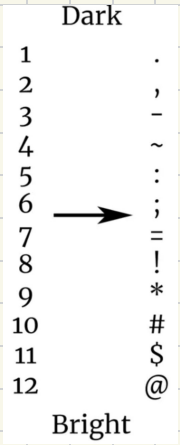
$$L = (N_x, N_y, N_z) \cdot (0, 1, -1)$$

↳ this yields not normalised. $L = [0, \sqrt{2}]$ since its

↳ from this



For ASCII character illuminance.



Variables

So now all that's left to do is to pick some values for R_1, R_2, K_1, and K_2. In the original donut code I chose R_1=1 and R_2=2, so it has the same geometry as my cross-section diagram above. K_1 controls the scale, which depends on our pixel resolution and is in fact different for x and y in the ASCII animation. K_2, the distance from the viewer to the donut, was chosen to be 5.