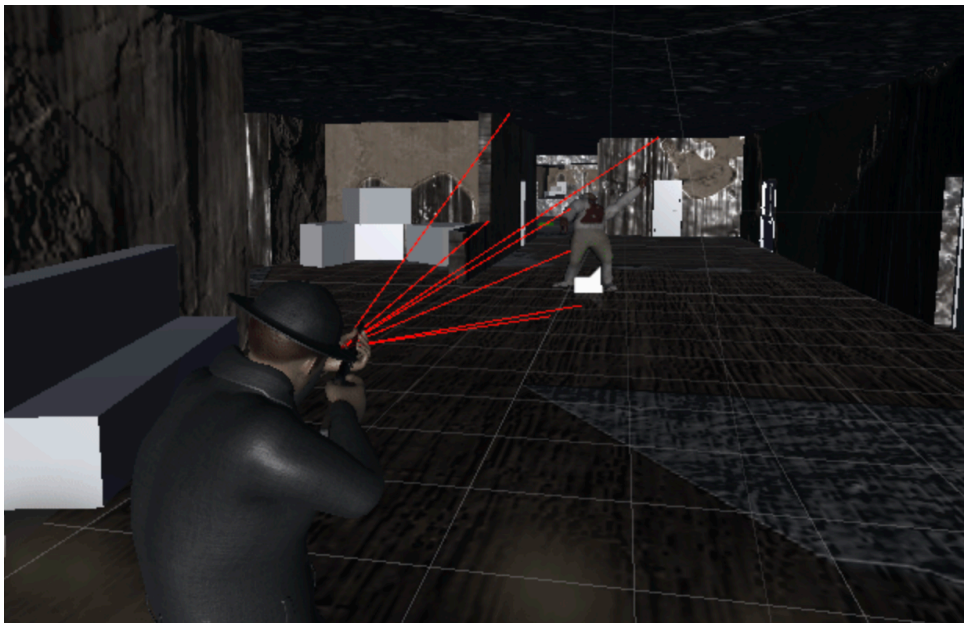
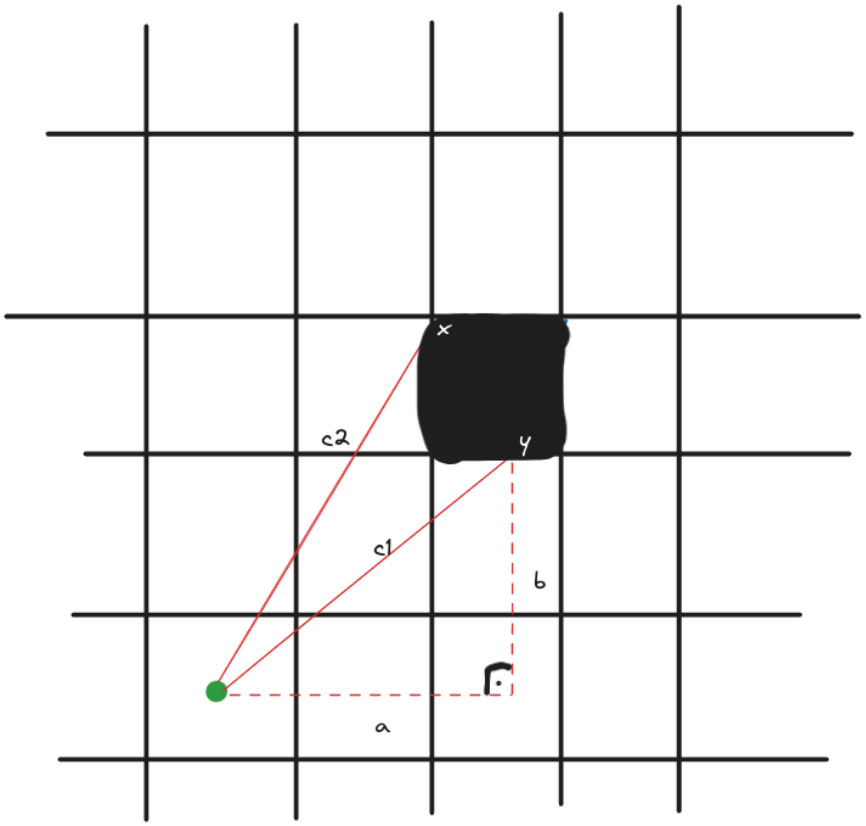
12 - cub3d

resources

- firstly, what tf is raycasting? 2d bir haritadan 3d bir görüntü elde etmeye yarayan bir tekniktir. belli bir noktadan (genelde oyuncunun göz hizası) çıkan ışınları kullanır. her bir ışın değdiği ilk noktayı tespit eder ve ona göre bir şey bastırılır. raytracing ise çok daha komplike bir 3D motorudur. raycastingden farkı değdiği ilk noktada durmaması böylece advanced-lightning ve shadowing tekniklerini hayata geçirebilmesidir.
- genel ışın mantıgı

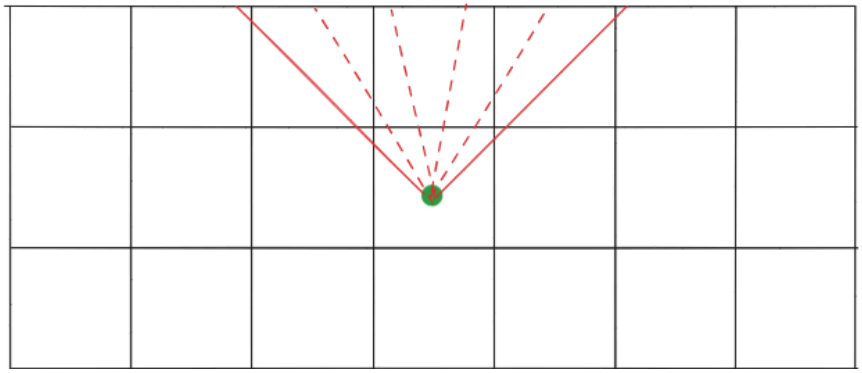


- en basit mantıkta raycasting şöyle çalışır:



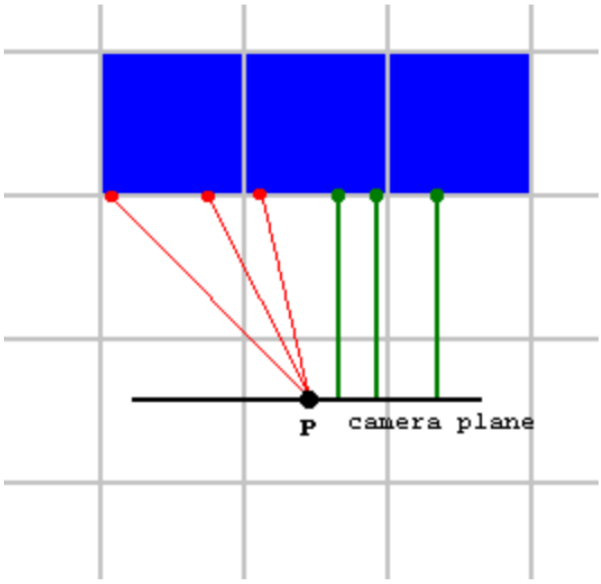
x ve koordinatlarını bildigimiz bir noktadan (bu noktada oyuncu bulunacak) haritada 1 olarak işaretlenmiş bir kareye doğru bakınca oyuncunun gözünden sürekli farazi bir ışın kümesi gönderilecek bu ışın kümesi duvara yani 1 gördügü bir yere carpana kadar sürekli arttırılacak. daha sonra çarptıgı nokta ile kendisi arasında bir pisagor hesaplaması uygulanacak. yani önce c2, sonra c1 hesaplanacak. ışınların çarptıgı noktalarda, c1 ve c2 sayılarıyla ters orantılı yükseklikte 3D bir çizgi çizilecek. yani c2 burada daha büyük oldugu icin daha kısa, c1 daha küçük oldugu icin daha uzun bir dik çizgi çizilecek. böylelikle oyuncuya daha yakın olan duvar (y) daha büyük (uzun), daha uzak olan (x) ise daha küçük (kısa) görünecek

- haritada her bir 1, 0 vs. bir kareyi temsil ediyor
- fish eye problem: raycasting çizdigi cizgi uzunluklarını pisagor teoremine göre yaptıgı icin düz duvara bakınca normalde bombeli görünmesi lazim. buna balık gözü problemi deniyor. mesela

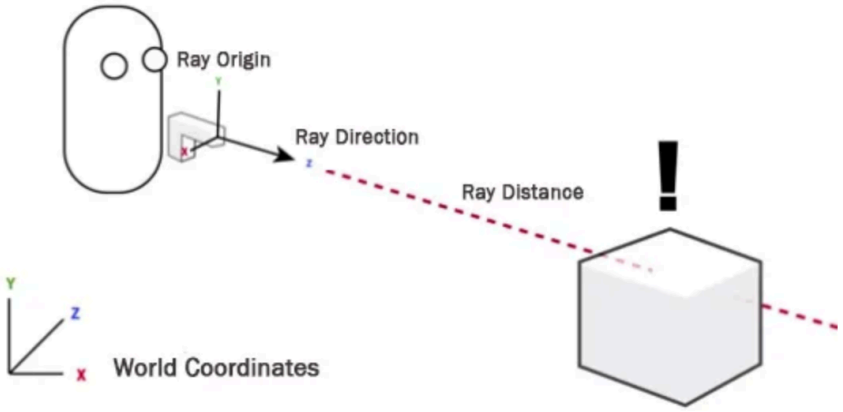


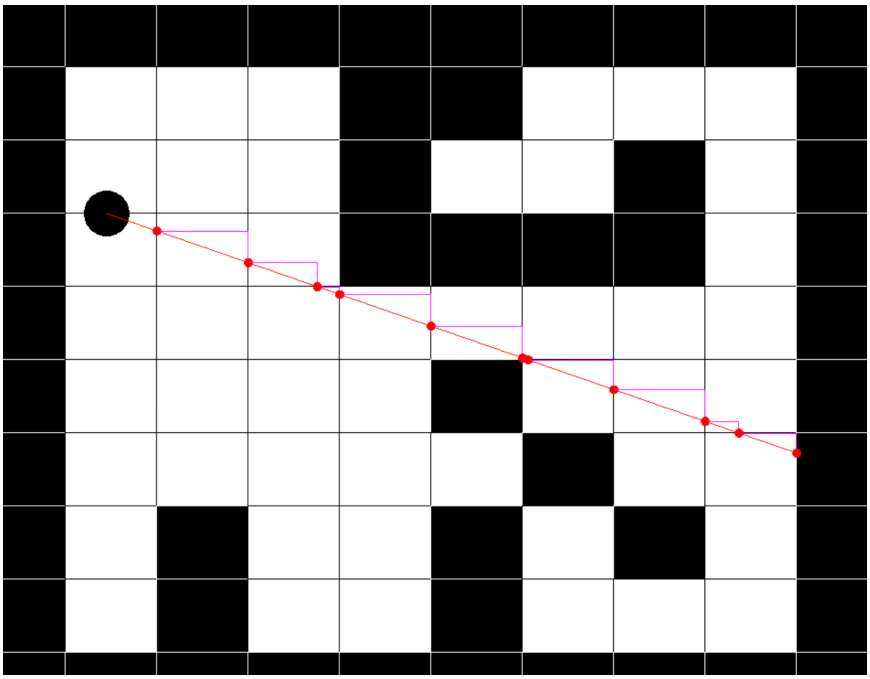
şekli 3D'de

olarak gözükebiliyor. bunun sebebi yine pisagorla alakalı. oyuncunun tam karşıtındaki duvar noktası sağ ve soldaki her hangi bir noktaya göre daha yakın oldugundan uçlara doğru çizilen her çizgi yanındakinden daha uzun çiziliyor bu da yumuşak bir kavis görünümü veriyor. bunun çözümü oyuncunun bakış açısıyla ışın açısı arasındaki farkı bu farkın kosinüsüyle çarpmak. bunu da camera plane yardımıyla yapiyoruz

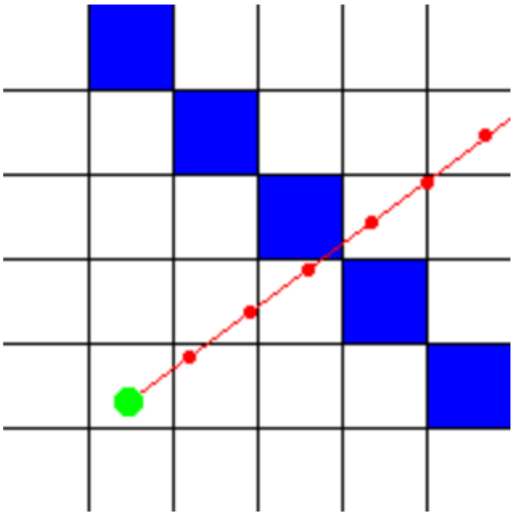


hesaplamaları player üzerinden pisagor teoremine göre degil de playerin bulundugu perspektiften yani camera plane'den direkt çizilen noktalara göre yapiyoruz böylece tam önümüzdeki düz duvarlar kavisli değil düz bir şekilde gözüküyor

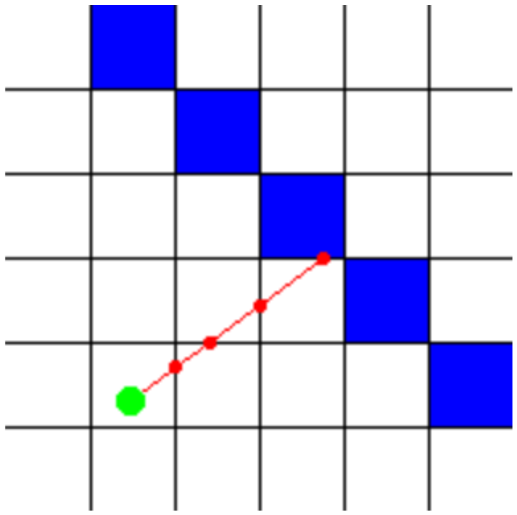




- raycasting ışını ilerlerken her seferinde belli bir constant ekleniyor ta ki duvar görene kadar ama o zaman da şöyle bir sorun olabilir:



normalde duvar olan bir yer tam arttırma boşluguna denk geldiginden duvar olarak görülmedi. bunun çözümü şu: raycast ışınını constant bir değerle arttırmayalim, onun yerine kendi doğrultusu boyunca, bir sonraki düşey ya da yatay herhangi bir çizgi görene kadar ilerletelim



böylece her zaman duvarları görecegimizden emin olabilir

- 2D'den 3D'ye geçerken üç farklı konsept kullanılacak
  - posX, posY → kullanıcının o anki (x, y) koordinatları
  - dirX, dirY → kullanıcının (x, y) direction yani baktıgı acı
  - planeX, planeY → kullanıcının bakış açısının, perspektifinin bilgisayar ekranına göre (x, y) koordinatları
-