Γραφικά Υπολογιστών

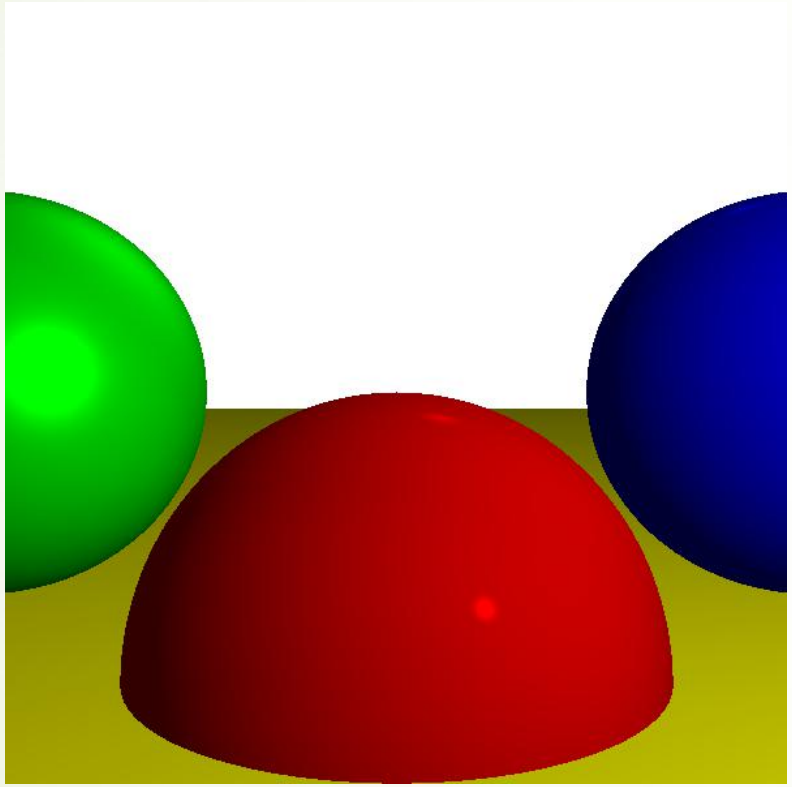
Μάθημα 6 - Σκιά και Αντανάκλαση

Γιώργος Σφήκας

Σήμερα θα μιλήσουμε...

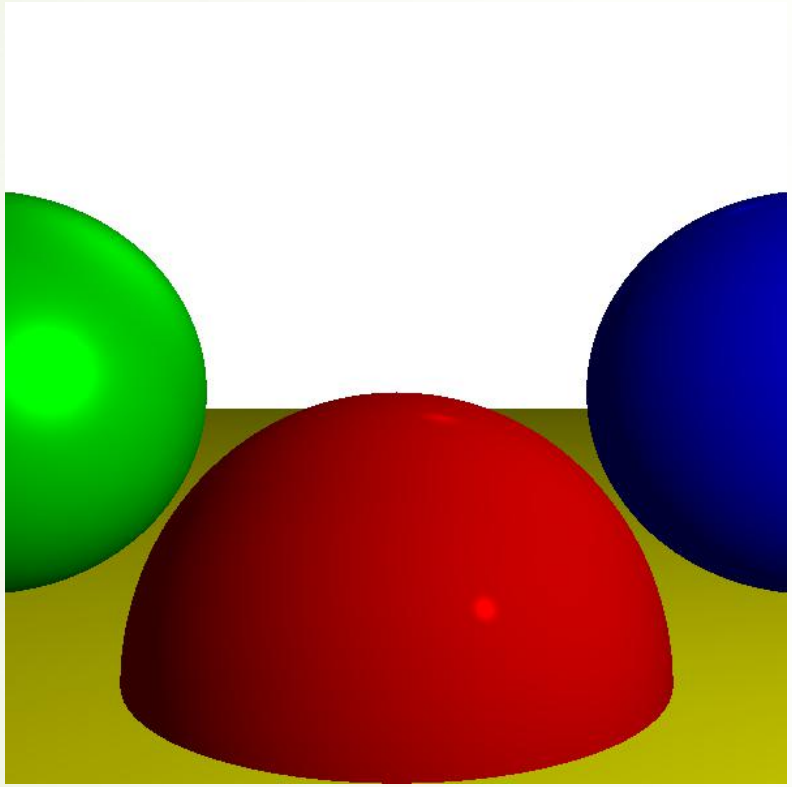
- Για δύο προσθήκες στον raytracer μας :
  - Τα αντικείμενα θα δημιουργούν σκιές
  - Τα αντικείμενα μπορούν να λειτουργούν και σαν καθρέφτες, δημιουργώντας αντανακλάσεις

Σκιά



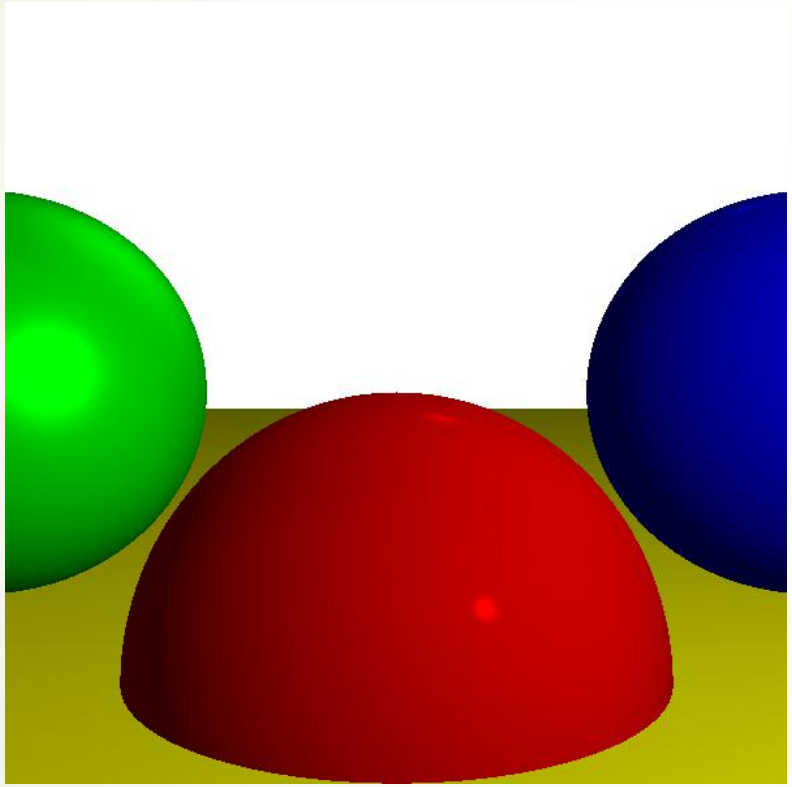
- Στο προηγούμενο μάθημα..
  - δημιουργήσαμε φωτεινές πηγές
  - δημιουργήσαμε αντικείμενα

Σκιά



- Στο προηγούμενο μάθημα..
  - δημιουργήσαμε φωτεινές πηγές
  - δημιουργήσαμε αντικείμενα
  - όπου έχουμε φως και αντικείμενα, έχουμε και σκιές...

Σκιά



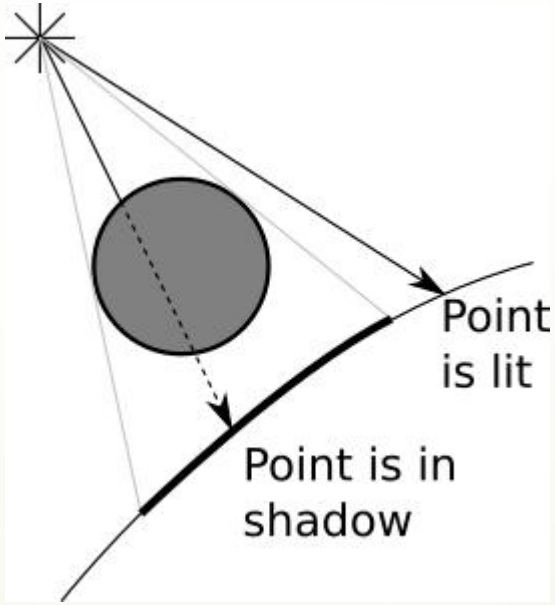
- Στο προηγούμενο μάθημα..
  - δημιουργήσαμε φωτεινές πηγές
  - δημιουργήσαμε αντικείμενα
  - όπου έχουμε φως και αντικείμενα, έχουμε και σκιές...
- Γιατί όμως δεν έχουμε σκιές;

Σκιά

- Πρέπει να κατανοήσουμε *πώς* δημιουργείται μια σκιά:

Σκιά

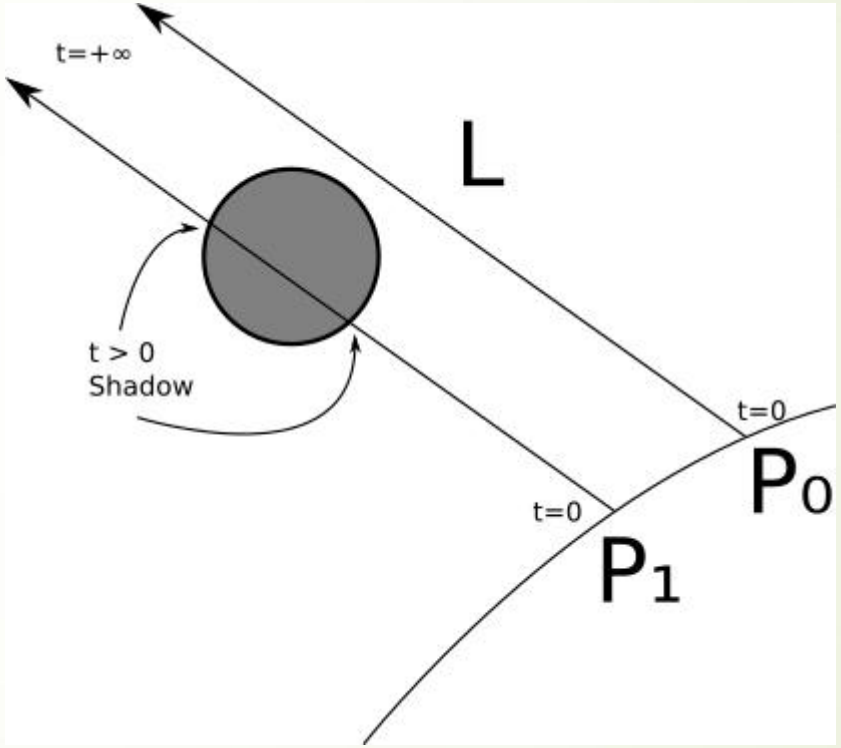
- Πρέπει να κατανοήσουμε *πώς* δημιουργείται μια σκιά:
- Οι σκιές δημιουργούνται όταν μεταξύ φωτεινής πηγής και αντικειμένου, υπάρχει άλλο αντικείμενο



Σκιά

Κατευθυνόμενο φως

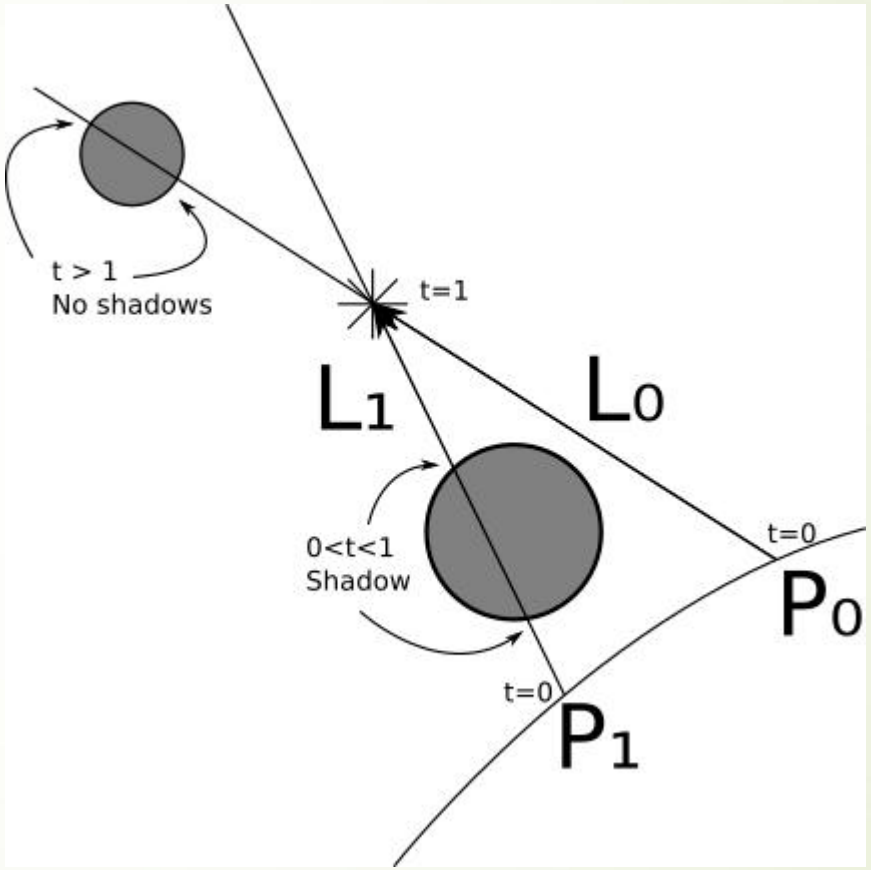
- Για το κατευθυνόμενο φως:
  - Πρέπει να κοιτάξουμε στην ευθεία $P + t\overrightarrow{L}$
  - Προσοχή στις αποδεκτές τιμές του t



Σκιά
Σημειακό φως

- Για το σημειακό φως:
  - Πρέπει να κοιτάξουμε στην ευθεία $P + t\overrightarrow{L}$
  - Προσοχή στις αποδεκτές τιμές του t

Προσοχή: Για t = 0 υπάρχει σίγουρα μια τομή που πρέπει να αγνοήσουμε..(ποιά;)



Υλοποίηση των σκιών

- Έχουμε ήδη γράψει κώδικα για να υπολογίσουμε την *κοντινότερη* τομή ακτίνας με σφαίρα...
- τον οργανώνουμε τώρα ως ξεχωριστή συνάρτηση

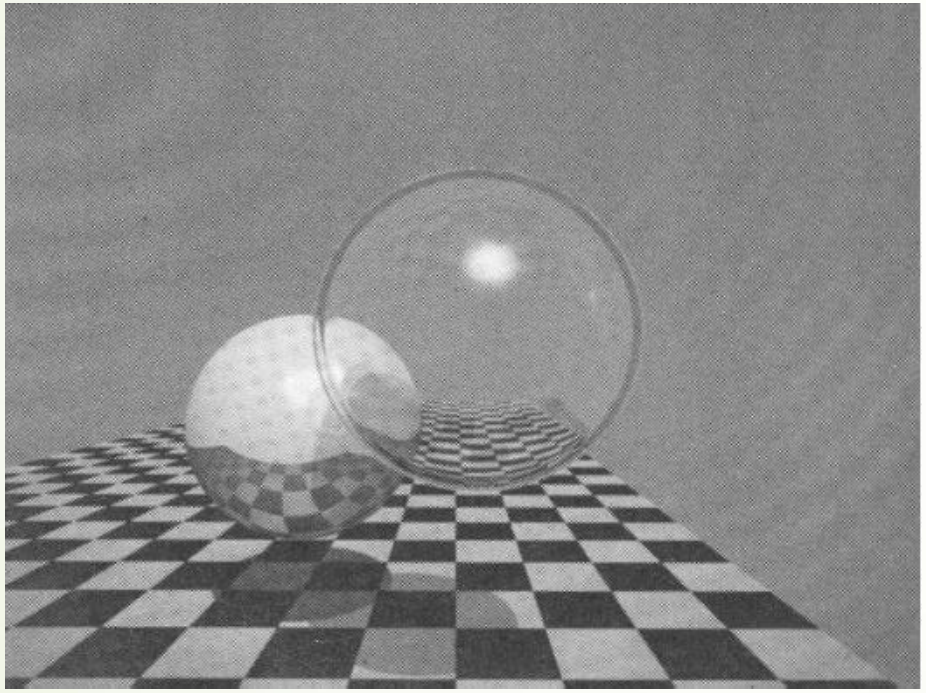
```
ClosestIntersection(O, D, t_min, t_max) {
    closest_t = inf
    closest_sphere = NULL
    for sphere in scene.Spheres {
        t1, t2 = IntersectRaySphere(O, D, sphere)
        if t1 in [t_min, t_max] and t1 < closest_t {
            closest_t = t1
            closest_sphere = sphere
        }
        if t2 in [t_min, t_max] and t2 < closest_t {
            closest_t = t2
            closest_sphere = sphere
        }
    }
    return closest_sphere, closest_t
}
```

```
TraceRay(O, D, t_min, t_max) {
    closest_sphere, closest_t = ClosestIntersection(O, D, t_min, t_max)
    if closest_sphere == NULL {
        return BACKGROUND_COLOR
    }
    P = O + closest_t * D
    N = P - closest_sphere.center
    N = N / length(N)
    return closest_sphere.color * ComputeLighting(P, N, -D, closest_sphere.specular)
}
```

Υλοποίηση των σκιών

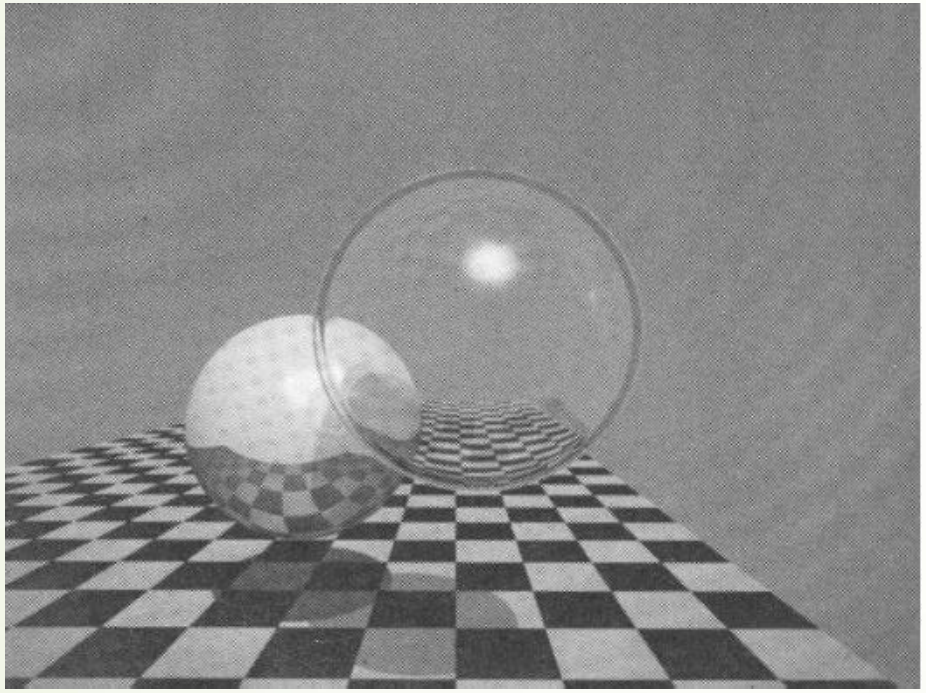
```
ComputeLighting(P, N, V, s) {
    i = 0.0
    for light in scene.Lights {
        if light.type == ambient {
            i += light.intensity
        } else {
            if light.type == point {
                L = light.position - P
                t_max = 1
            } else {
                L = light.direction
                t_max = inf
            }

            // Shadow check
         ❶ shadow_sphere, shadow_t = ClosestIntersection(P, L, 0.001, t_max)
            if shadow_sphere != NULL {
                continue
            }

            // Diffuse
            n_dot_l = dot(N, L)
            if n_dot_l > 0 {
                i += light.intensity * n_dot_l / (length(N) * length(L))
            }

            // Specular
            if s != -1 {
                R = 2 * N * dot(N, L) - L
                r_dot_v = dot(R, V)
                if r_dot_v > 0 {
                    i += light.intensity * pow(r_dot_v / (length(R) * length(V)), s)
                }
            }
        }
    }
    return i
}
```

Αντανάκλαση



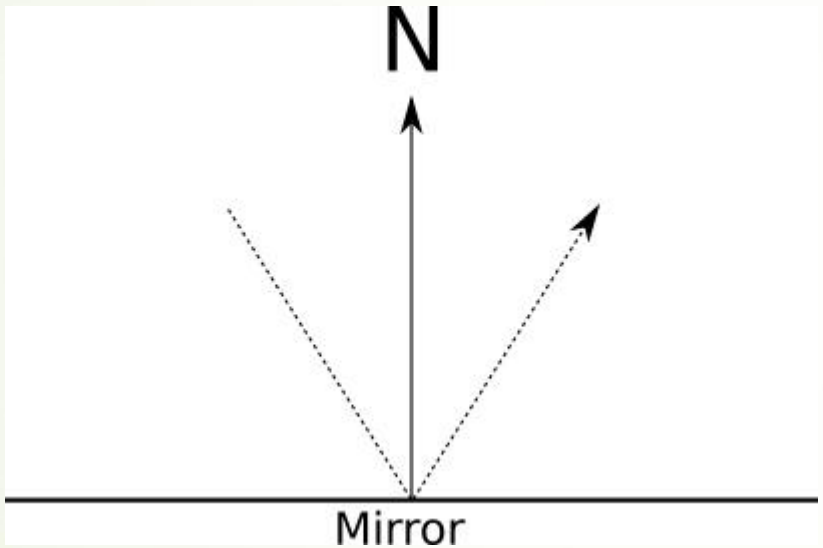
- Γυαλιστερά αντικείμενα = είπαμε ότι μοιάζουν κατά ένα συγκεκριμένο τρόπο με καθρέφτες..
- Δεν είναι ακριβώς όμως καθρέφτες.

Αντανάκλαση



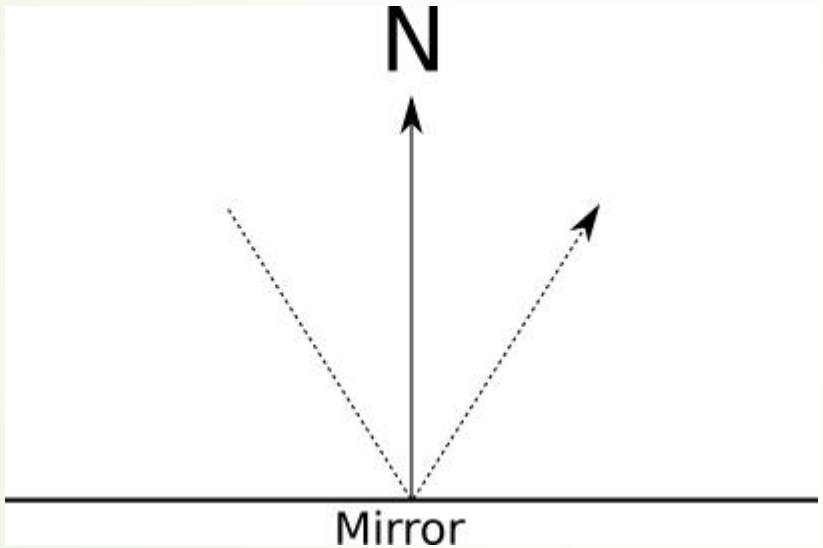
- Η υλοποίηση της συμπεριφοράς ενός καθρέφτη είναι τρομερά απλή σε έναν raytracer !!
- Ήταν ένα βασικό “selling point” του raytracing από το 1980 (Whitted, «An improved illumination model for shaded display»)

Πώς “λειτουργεί” ένας καθρέφτης;



- Έστω ότι παρακολουθώντας μία ακτίνα, υπολογίζουμε ότι τέμνεται με ένα αντικείμενο που είναι καθρέφτης. Τι χρώμα πρέπει να επιστρέψουμε ως αποτέλεσμα;

Πώς “λειτουργεί” ένας καθρέφτης;



- Έστω ότι παρακολουθώντας μία ακτίνα, υπολογίζουμε ότι τέμνεται με ένα αντικείμενο που είναι καθρέφτης. Τι χρώμα πρέπει να επιστρέψουμε ως αποτέλεσμα;
- **Απάντηση**: Το χρώμα που αντιστοιχεί σε μια *νέα* ακτίνα, κατά την κατεύθυνσης ανάκλασης (R)

Δηλαδή η συνάρτηση traceRay θα καλέσει .. τη συνάρτηση traceRay !

«Reflectiveness»

- Θα θεωρήσουμε όχι μόνο τις δύο ακραίες περιπτώσεις:
  - Ένα αντικείμενο είναι καθρέφτης
  - Ένα αντικείμενο δεν είναι καθρέφτης
- Αλλά εύκολα μπορούμε να υλοποιήσουμε και την “ενδιάμεση” κατάσταση που ένα αντικείμενο έχει δικό του χρώμα αλλά επίσης λειτουργεί και σαν καθρέφτης (ζυγισμένος μέσος όρος με τη νέα ακτίνα)
  - Reflectiveness = Το βάρος στον μ.ο. -- θα είναι ιδιότητα του αντικειμένου

Υλοποιώντας την αντανάκλαση

```
sphere {
    center = (0, -1, 3)
    radius = 1
    color = (255, 0, 0)  # Red
    specular = 500  # Shiny
    reflective = 0.2  # A bit reflective
}
sphere {
    center = (-2, 0, 4)
    radius = 1
    color = (0, 0, 255)  # Blue
    specular = 500  # Shiny
    reflective = 0.3  # A bit more reflective
}
sphere {
    center = (2, 0, 4)
    radius = 1
    color = (0, 255, 0)  # Green
    specular = 10  # Somewhat shiny
    reflective = 0.4  # Even more reflective
}
sphere {
    color = (255, 255, 0)  # Yellow
    center = (0, -5001, 0)
    radius = 5000
    specular = 1000  # Very shiny
    reflective = 0.5  # Half reflective
}
```

Υλοποιώντας την αντανάκλαση

```
ReflectRay(R, N) {
    return 2 * N * dot(N, R) - R;
}
```

```
color = TraceRay(O, D, 1, inf, recursion_depth)
```

```
TraceRay(O, D, t_min, t_max, recursion_depth) {
    closest_sphere, closest_t = ClosestIntersection(O, D, t_min, t_max)

    if closest_sphere == NULL {
        return BACKGROUND_COLOR
    }

    // Compute local color
    P = O + closest_t * D
    N = P - closest_sphere.center
    N = N / length(N)
    local_color = closest_sphere.color * ComputeLighting(P, N, -D, closest_sphere.specular)

    // If we hit the recursion limit or the object is not reflective, we're done
 ❶ r = closest_sphere.reflective
    if recursion_depth <= 0 or r <= 0 {
        return local_color
    }

    // Compute the reflected color
    R = ReflectRay(-D, N)
 ❷ reflected_color = TraceRay(P, R, 0.001, inf, recursion_depth - 1)

   ❸return local_color * (1 - r) + reflected_color * r
}
```

Ανακεφαλαίωση

- Έχουμε μέχρι στιγμής υλοποιήσει, στο πλαίσιο του αλγόριθμου raytracing (παρακολούθηση ακτίνας)
  - Point lights, Directional lights, ambient lights
  - Diffuse, specular lighting
  - Shadows
  - Reflections

Μελέτη στο βιβλίο - πηγές

- Κεφάλαιο 4, https://gabrielgambetta.com/computer-graphics-from-scratch/04-shadows-and-reflections.html

Την επόμενη φορά..

- Άλλους τρόπους να επεκτείνουμε τον raytracer μας, μεταξύ των οποίων:
  - Η κάμερα όχι απαραίτητα στην αρχή των αξόνων
  - Διάθλαση φωτός
  - Αντικείμενα διαφορετικά από σφαίρες
  - ..

Ερωτήσεις ... ;;;