**שלב 7 - הוספת הצללה, שקיפות והשתקפות, השפעת שקיפות על הצללה (טיוטה)**

לפני הכל נקפיא את השלב הקודם באתר **github** ע"י יצירת **branch** חדש בשם **recitation6**. לאחר מכן נוודא שענף ברירת המחדל נשאר **master**.

שלב זה ניתן לשבועיים – אך אין לדחות את תחילת העבודה עליו...

***נ.ב.1*** *חובה להקפיד על תיעוד בפורמט* ***javadoc*** *לכל המחלקות והפעולות החדשות ולעבוד ע"פ הסדר כפי שנלמד בקור*

**חלק 1 - הצללה**

1. במחלקת **Render**:
   1. נוסיף קבוע עבור גודל הזזת ראשית קרניים עבור קרני הצללה, שקיפות והשתקפות (ניתן להקטין את הערך שלו בהתאם לסדרי גודל הצורות בתמונה שלכם כך שההזזה לא תהיה ניכרת לעין בתמונה):

**private** **static** **final** **double** ***DELTA*** = 0.1;

* 1. נוסיף פעולת בדיקת אי הצללה בין נקודה למקור האור שחתימתה (ניתן להוסיף פרמטרים אם אתם מוצאים שזה יועיל מבחינת הביצועים):

**private** **boolean** unshaded(Vector l, Vector n, GeoPoint gp)

* 1. כמובן לא נשכח להוסיף תיעוד בפורמט **JavaDoc** גם עבור הקבוע וגם עבור הפעולה הנ"ל
  2. נוסיף את בדיקת ההצללה בפונקציה **calcColor** בהתאם למה שנלמד בכיתה

1. נבצע את כל הטסטים שישנים ונוודא שהם עברו ושהתמונות לא השתבשו
2. נגלתה תקלה מכיוון בתמונת תאורה על פני זוג משולשים מופיע צל מהמשולש שנמצא מאחורי מקור התאורה. על מנת לתקן את הבעיה נעשה **refactoring**:
   1. נוסיף בפונקציה **unshaded** פרמטר מסוג **LightSource** ובזימון הפונקציה מעביר את מקור התאורה הנוכחי לפונקציה
   2. בממשק **LightSource** נוסיף חתימת פונקציה חדשה:

**double** getDistance(Point3D point)

* 1. נממש את הפעולה עבור שלושת מקורות התאורה המממשים את הממשק

**נ.ב.** במקור אור כיווני (**DirectionalLight**) נחזיר ערך **Double.POSITIVE\_INFINITY**

* 1. עכשיו "נסנן" את נקודות החיתוך באחד משני הדרכים:
     1. **דרך 1**: בפונקציה **unshaded** עם רשימת החיתוכים שהתקבלה אינה ריקה – נספור את כל החיתוכים שקרובים לראש הקרן מאשר מרחק בין הנקודה לבין מקור האור
     2. **דרך 2**:
        1. נשנה את הפונקציות בממשק **Intersectable** כדלקמן (כמובן עליכם להשלים את כל התיעוד בפורמט **Javadoc** בהתאם):

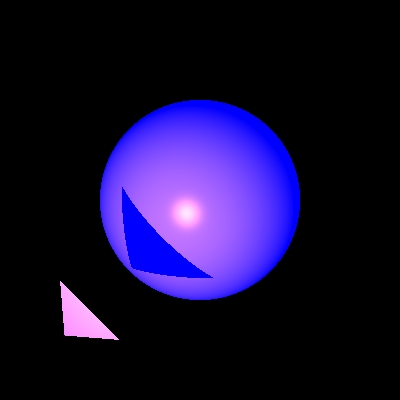
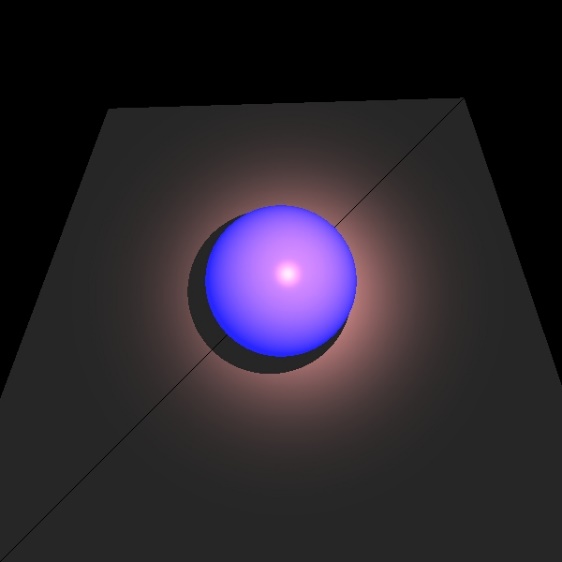
**default** List<GeoPoint> findIntersections(Ray ray) {

**return** findIntersections(ray, Double.***POSITIVE\_INFINITY***);

}  
List<GeoPoint> findIntersections(Ray ray, **double** max);

* + - 1. נשנה בכל המחלקות ששם ממומשת הפונקציה המקורית:
         1. נוסיף פרמטר חדש כנ"ל
         2. נעדכן את הקוד של הפונקציה כך שתתחשב במרחק המירבי עד לנקודת החיתוך (לא תכניס נקודות שרחוקות יותר מערך הפרמטר – כמובן שהבדיקה תתבצע כדוגמה הבאה: **alignZero(t – max) <= 0**)
      2. בפונקציה **unshaded** נוסיף לזימון של **findIntersections** מרחק בין הנקודה למקור האור כארגומנט עבור הפרמטר החדש.
      3. אין צורך בשינויים נוספים
  1. נריץ מחדש את הטסטים הישנים ונוודא שהבעיה תוקנה

1. נוריד מתיבת ההגשה של התרגיל את המודול **ShadowTests.java**, נשים אותו בחבילת **unittests** ונריץ אותו – נבדוק את התמונות:

** **

1. בטסט הזה ישנן 5 (ארבע) פונקציות של בדיקת הצללה של משולש על ספירה וכולן מיצרות תמונה זהה. עליכם לעדכן את הפרמטרים בכל הפעולות האלה חוץ מהראשונה על מנת לקבל תמונות עם הזזת המשולש והזזת מקור האור כפי שמופיע במצגת המעבדה

**חלק 2 – שקיפות והתשקפות**

1. נוסיף במחלקה **Material** שני שדות חדשים מסוג **double** בשם **\_kT** (מקדם שקיפות) ו-**\_kR** (מקדם השתקפות).
   1. נוסיף פעולות מאחזרת (גטרים) עבור השדות החדשים (עם תיעוד מתאים בפורמט **javadoc**)
   2. נוסיף בנאי חדש עם פרמטרים עבור כל השדות כולל השדות החדשים, בבנאי הישן נזמן את הבנאי החדש עם ערכי 0 עבור השדות החדשים
2. במחלקת **Render**:
   1. נוסיף שני קבועים עבור תנאי עצירה ברקורסיה של שקיפויות\התשקפויות (ניתן לשנות את הערכים ע"פ הערכתכם והבנתכם שתצטרכו להסביר בהצגת השלב כמובן):

**private** **static** **final** **int** ***MAX\_CALC\_COLOR\_LEVEL*** = 10;

**private** **static** **final** **double** ***MIN\_CALC\_COLOR\_K*** = 0.001;

* 1. נוסיף פעולות חישוב קרן השתקפות וקרן שקיפות (כמובן עם תיעוד בפורמט **javadoc**) – בקרניים האלה תוזז נקודת תחילת הקרן על ישר הנורמל לגאומטריה בכיוון הקרן החדשה – בדומה למה שעשינו בבניית קרני הצללה
  2. מכיון שעכשיו נצטרך לחשב את החיתוך הקרוב ביותר ממקום אחד – נעשה **refactoring** ע"י בניית פעולה שתחשב את החיתוכים וגם תחשב את החיתוך הקרוב לראשית הקרן עם החתימה הבאה (כמובן עם תיעוד בפורמט **javadoc**), הפעולה תחזיר **null** אם לא היו נקודות חיתוך לקרן:

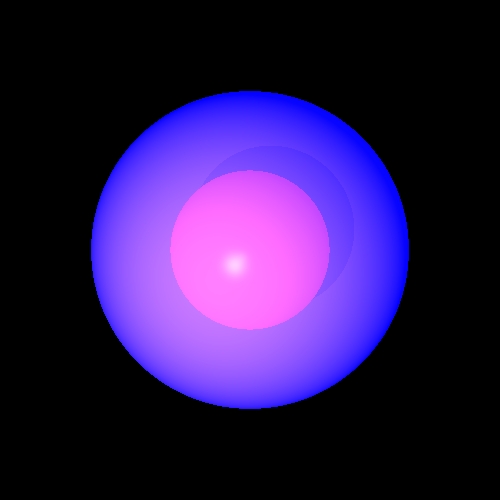
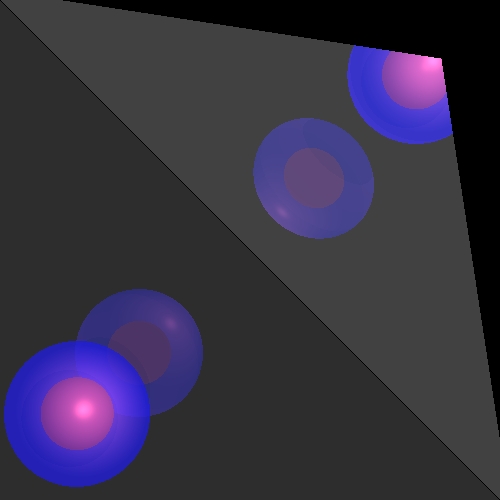
**private** GeoPoint findCLosestIntersection(Ray ray)

* 1. שעשה עדכון בקוד של פעולת הרינדור כך שתשתמש בפעולה הזו החדשה
  2. נוסיף פעולה חדשה **calcColor** עם החתימה הבאה (כמובן עם תיעוד בפורמט **javadoc** – התיעוד יהיה מהפעולה המקורית עם עדכונים בהתאם) – הפעולה תזמן פעולת עזר כפי שנלמד בכיתה (ראו בסעיף הבא):

**private** Color calcColor(GeoPoint gp, Ray ray)

* 1. נשנה את הפעולה המקורית **calcColor** כך שתהפוך לפעולת עזר רקורסיבית בהתאם למה שנלמד בכיתה (שימו לב שהוספת רכיב האורה הסביבתית יעבור לפעולה "הראשית" הנ"ל) – עם פרמטרים נוספים ועם תמיכה בשקיפות והשתקפות, כפי שנלמד בכיתה, כולל פעולות חדשות נוספות (עם תיעוד בפורמט **javadoc**)
  2. נעדכן את הפונקציה **unshaded** כך שרק אוביקטים ללא מקדם שקיפות (זאת אומרת מקדם שקיפות = 0) יגרמו להצללה (בחלק הזה) כפי שנלמד בכיתה
  3. נבצע את כל הטסטים הישנים ונוודא שהם עברו ושהתמונות לא השתבשו

1. נוסיף לחבילה **unittests** מודול **ReflectionRefractionTests.java** מתיבת ההגשה – המודול כולל את שני הטסטים שהוצגו בכיתה (שני כדורים מקוננים לבד, שני כדורים מקוננים לפני שני משולשים של מראה, בינתיים נתעלם מהתמונה השלישית שנוצרת במודול הזה):

1. נשים לב על כך שבניית קרן עם הזזת נקודה – זו פעולה שחוזרת על עצמה כבר שלוש פעמים, לכן כדאי להוריד אותה לפונקציה נפרדת. מיקום הפונקציה ע"פ **RDD** הוא מחלקת הקרן. מכיוון שמדובר בבניית קרן חדשה (ע"פ נקודה מקורית, כיוון הקרן, וקטור הנורמל (שעל הישר שהוא מגדיר יש להזיז את הנקודה של ראש הקרן)) – נוסיף את הפונקציה כבנאי נוסף של קרן ונעדכן בכל המקומות הרלוונטיים את הקוד כך שישתמש בבנאי החדש של הקרן. כמובן הקבוע **DELTA** שהגדרנו קודם יועבר למחלקת **Ray**. נריץ את כל הטסטים ונוודא את תקינות תוצאותיהם.

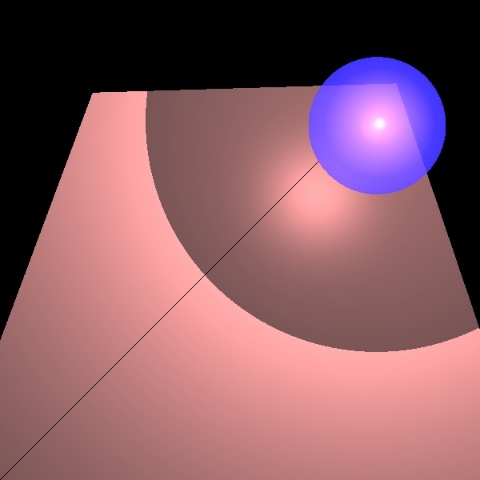
**חלק 3 – שקיפות והצללה חלקית**

נרצה לקבל **הצללה חלקית** במקרה שהגוף או הגופים שחסומים את מקור האור מהנקודה הינם בעלי שקיפות ברמה כזו או אחרת. על מנת להשיג מטרה זו:

1. נעתיק את הפונקציה **unshaded** ובעתק נשנה את השם ואת סוג הערך המוחזר כדלקמן:

**private** **doulbe** transparency(LightSource ls, Vector l, Vector n, GeoPoint geopoint)

1. נוסיף לפונקציה תיעוד בפורמט **javadoc** ונשנה את המימוש של הפעולה כפי שנלמד בכיתה ומפיע במצגת המעבדה
2. בפונקציה **calcColor** נחליף את הקריאה ל-**unshaded** ונשתמש בפונקציה החדש בהתאם, כפי שנלמד בכיתה ומופיע במצגת המעבדה
3. שימו לב: הקוד במצגת הינו פסיודו-קוד ולא חייב להיות תקין ב-100% מבחינת תחביר java
4. נריץ את **כל** הטסטים מחדש ונעבור על כל התמונות – נוודא שהכל תקין והתמונה האחרונה החדשה נראית כמו:



1. נבנה תמונה משלנו (חובה!) שמשלבת 3-4 מספר גופים שונים שמאפשרת להראות את כל האפקטים החדשים ביחד
2. נ.ב. אל תיצרו אובייקטים סגורים (למשל כדור) בעלי שקיפות והשתקפות בו זמנית!!! אחרת בניית התמונה תיקח לכם זמן רב בגלל הקרניים שנלכדות ומטיילות בתוך הכדור

**חלק לבונוסים**

ניתן לקבל בונוס של עד 3 נק' ע"י (1 נק' עבור כל אחת מהסעיפים):

* בניית תמונה הכוללת גופים רבים (10 או יותר) גופי **שונים** (נכל הסוגים שמימשתם!) תוך הדגמת כל התכונות ואפקטים שבוצעו עד כה, כולל השלב הזה – לא תתקבל תמונה עם אותו אחד או אותם שני אוביקטים שמוקמו בצורה רנדומלית – התמונה חייבת להיות מרשימה.
* הצגת התמונה מהבונוס הראשון מזויות שונות וממרחקים שונים (ע"י הזזת הקמרה). לא תתקלנה תמונות של סתם קירוב הקמרה על אותו ציר כמו במיקום אחד.
* השלמת כל החלקים והבונוסים (אם החלטת לעשות אותם) תוך שבוע ממתן המטלה בקבוצתך

בהצלחה!