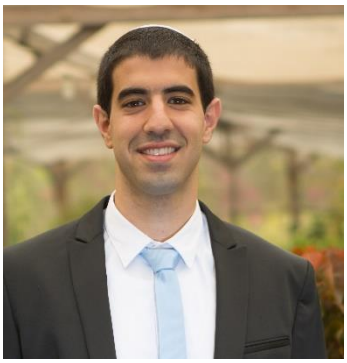
24/6/2019

מיני פרויקט-מבוא להנדסת תוכנה

פרויקט זה הוא היכרות עם הנדסת תוכנה, עיצוב וארכיטקטורה



אורי ביבי- 204248140
גולן מעטוף -311188585

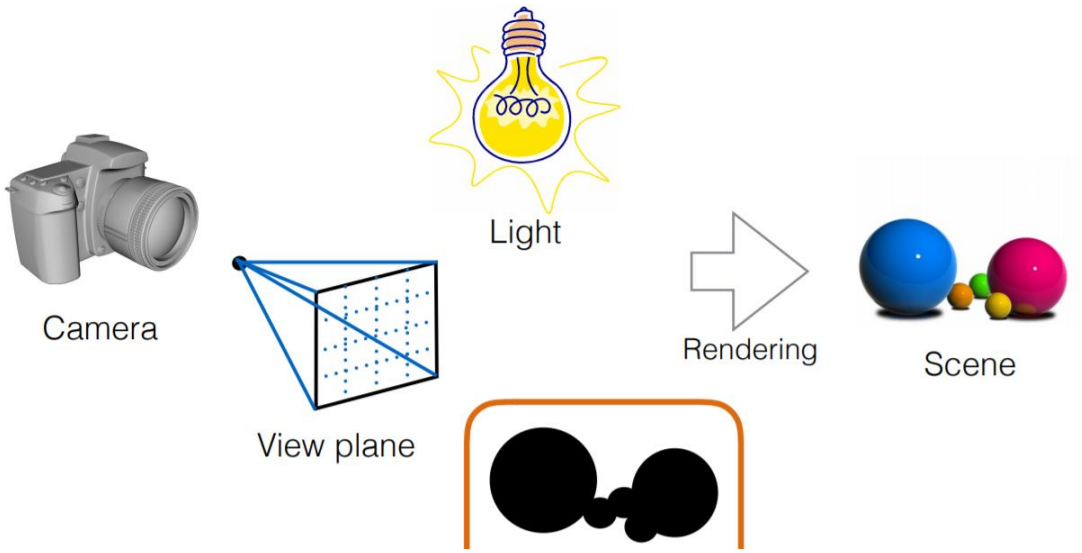
מטרת המיני הפרוייקט הוא לממש את עקרונות הנדסת תוכנה OOP (object oriented programing-תכנות מונחה עצמים) הכוללת כימוס, ירושה ופולימורפיזם.

יישום תכונות מפתח של oop כגון:reuse של קוד במקום להגדיר אותו מחדש כמה פעמים,

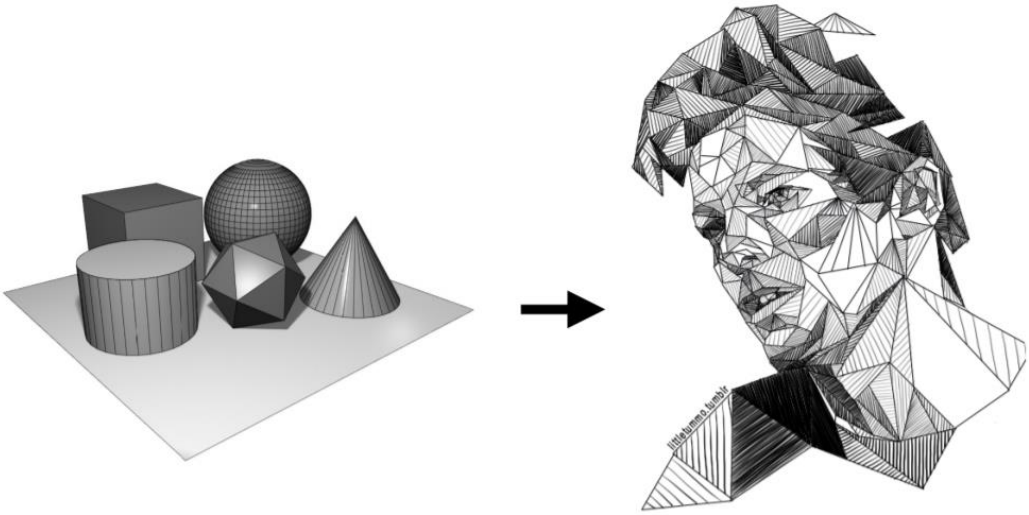
"הפרד ומשול"-חלוקה של התוכנית הגדולה לתתי משימות שניתן למצוא להם פיתרון וכך לפתור את הבעיה הגדולה, אירגון קוד אסתטי- לפי עיקרון ה-agile תוכנית צריכה להיות מתועדת ע"י הקוד עצמו. הקוד צריך להיות ברור מספיק בשביל להבין את התוכנית,

ו-debugging- יכולת לאתר את הבעיות ולאחר מכן פתירתן.

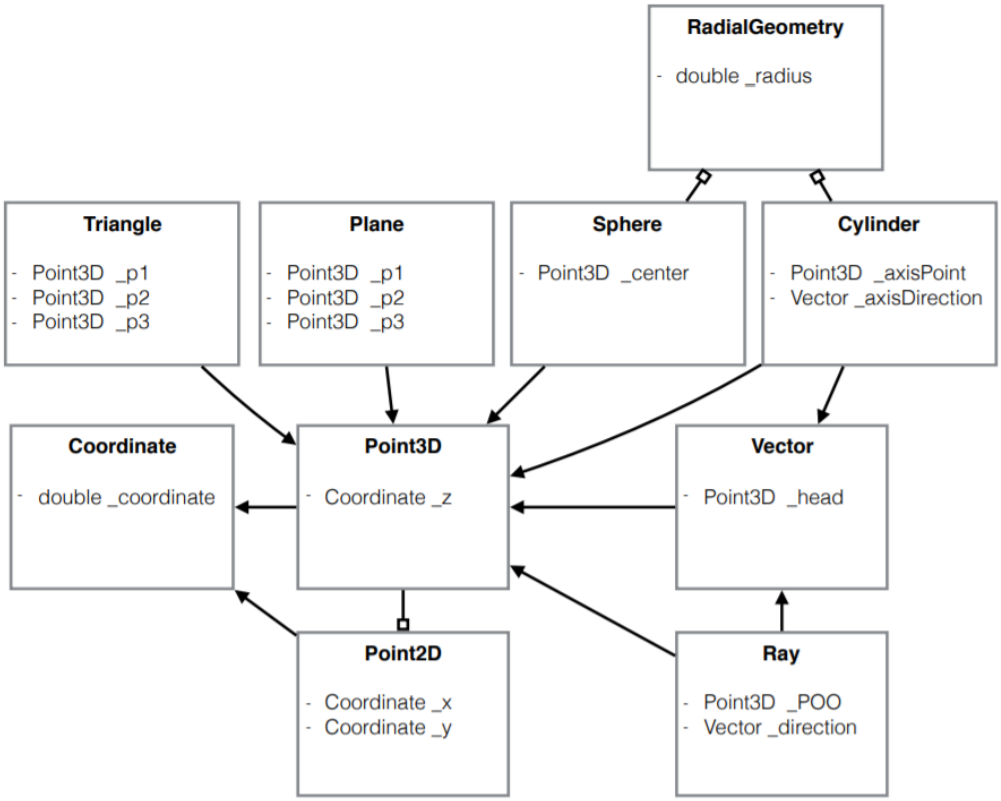
נביא את העקרונות הללו לידי ביטוי ע"י יצירת סצנה ממחושבת המורכבת מאלמנטים(מצלמה,אור) וצורות מורכבות שמפגש קרני האור בצורות ומיקומם ביחס למצלמה ייצרו לנו סצנה.



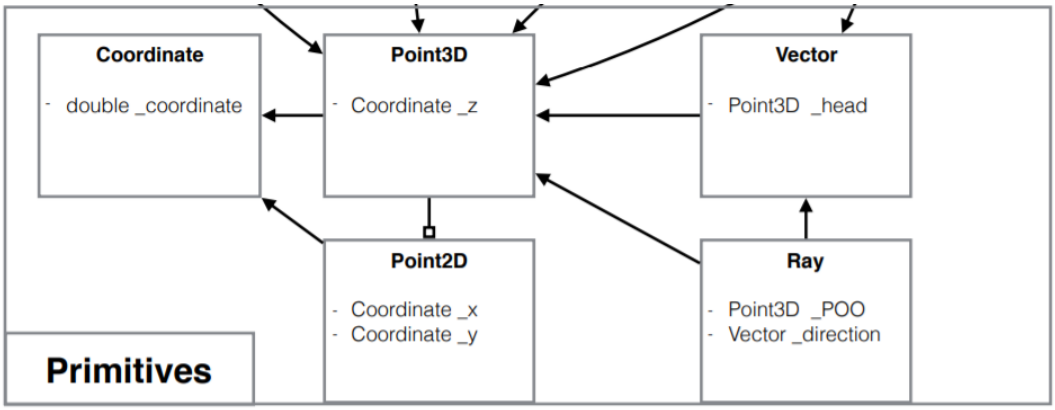
הצורות המורכבות יהיו מורכבות מצורות פרימיטיביות שקל ליצור. אנו נפרק את הסצנה לתתי צורות פשוטות וכך נפתור את הבעיה.("הפרד ומשול")



פירוט המחלקות הוא כדלהלן ויחסי הגומלין ביניהם כדלהלן:



חבילת הפרימיטיבים

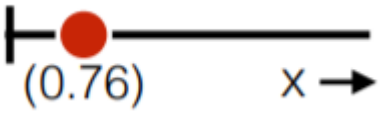


מכילה את המחלקות הבסיסיים מהם אנו בונים את המחלקות המורכבות.

מחלקת Utility

מחלקת עזר עבור מחלקת Coordinate בשביל לבצע קירובים.

מחלקת Coordinate:



מכילה משתנה אחד המכיל ערך מסוג double המייצג מרחק מראשית הצירים בציר כלשהו.

מכילה בנאים, פונקציות gettets ו-setters .

פונקציית equals המשווה בין האובייקט שממנו נראה הפונקציה לאובייקט אחר.

פונקצית toString המחזירה פירוט על שדות המחלקה.

פונקציית substract המחסירה בין הקואורדינטה לקואורידנטה המתקבלת כארגומנט ומחסירה את הקואורידנטה החדשה.

פונקציית add המוסיפה את ערך הקואורידנטה המתקבלת כארגומנט לקואורידנטה ומחזירה את הערך החדש.

פונקציית multiply המכפילה בין הקואורינטה המתקבלת כארגומנט לקואורדינטה ומחזירה את הערך החדש.

פונקציית scale המכפילה בין ערך double כלשהו לערך הקואורדינטה ומחזירה את הערך החדש.

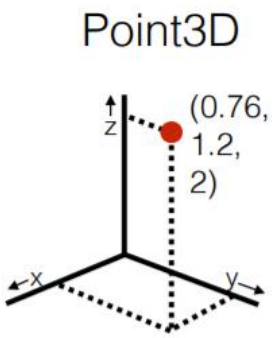
מחלקת Point2D

מכילה 2 שדות של קואורידנטות. אחד עבור ציר X ואחד עור ציר Y.

מכילה בנאים ופונקציות getters ו-setters.

פונקציית equals המשווה בין אובייקט מסוג מחלקה זו לאובייקט אחר.

מחלקת Point3D



מכילה 3 שדות של קואורידנטות. אחד עבור ציר X השני עבור ציר Y והשלישי עבור ציר Z.

מכילה בנאים ופונקציות getters ו-setters.

פונקציית equals המשווה בין אובייקט ממחלקה זו לאובייקט אחר.

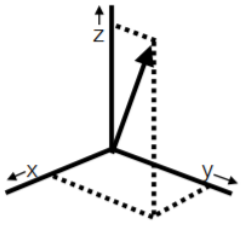
פונקציית toString המחזירה פירוט על שדות המחלקה.

פונקציית add המוסיפה את ערכי הקואורדינטות של נקודה המתקבלת כפרמטר לנקודה אחרת.

פונקציית substract המחסירה את ערכי הקואורדינטות של נקודה המתקבלת כפרמטר לנקודה אחרת.

פונקציית distance המחשבת ומחזירה את המרחק בין 2 נקודות תלת מימד.

מחלקת Vector



מכילה שדה של קואורדינטה אחת המייצגת את ראש הווקטור המסתיים בראשית הצירים,וכך נתאר קו ישר.

מכילה בנאים ופונקציות getters ו-setters.

פונקציית equals המשווה בין אובייקט ממחלקה זו לאובייקט אחר.

פונקציית toString המחזירה פירוט על שדות המחלקה.

פונקציית add המוסיפה את ערכי הקואורדינטות של ראש הווקטור המתקבל כפרמטר לראש הווקטור. כלומר מחבר בין ווקטורים.

פונקציית substract המחסירה בין קואורדינטות.

פונקציית scale הכפילה בין ערך double המתקבל כפרמטר לווקטור ומשנה את הווקטור לערכו החדש.

פונקציית crossProduct המבצעת מכפלה וקטורית בין וקטורים,ומחזירה את הווקטור המאונך אליהם. אם התוצאה שווה ל-0,אזי הווקטורים מקבילים ונזרוק חריגה.

כמתואר בתרשים הבא:

$$u \times v = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \times \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = \begin{pmatrix} u_2 v_3 - u_3 v_2 \\ u_3 v_1 - u_1 v_3 \\ u_1 v_2 - u_2 v_1 \end{pmatrix}$$

פונקציית length מחזירה את האורך של הווקטור, כמתואר בתרשים הבא:

$$|v| = \sqrt{x^2 + y^2 + z^2}$$

פונקציית normalize המשנה את ערך הוקטור שקרא לו לערך מנורמל, כלומר שאורכו של הוקטור הוא 1 ע"י חילוק כל קואורדינטה שלו באורך שקיבלנו בפונקצייה length(כלומר באורך שלו).

פונקצייה dotProduct המחזירה ערך double ע"י מכפלה סקלרית בין 2 וקטורים. אם הערך יהיה 0 אזי הוקטורים מאונכים זה לזה.

$$u \cdot v = u_1 v_1 + u_2 v_2 + u_3 + v_3$$

החישוב מתואר בתרשים הבא:

<u>מחלקת Ray</u>

מכילה 2 שדות:האחד הוא הנקודה במרחב ממנה מתחיל הווקטור והשני הוא הווקטור.

מכילה בנאים ופונקציות getters ו-setters.

חבילת הטסטים(Tests):

על מנת לעשות טסטים ולבחון את תקינות הקוד נבנה דוגמאות הרצה על המחלקות שבנינו.

מכילה טסטים על פונקציות ממחלקות Point3D ו-Vector.

הצורות הגיאומטריות, האלמנטים, סצנות ורנדור ראשוני– שלב 2

כעת, אחרי שיש לנו את כל הפרימיטיביים שיאפשרו לנו ליצור צורות במרחב נוכל להתחיל ליצור את הצורות ואף לראות תמונה בסיום השלב. לפני שנכתוב את המחלקות של הצורות הגיאומטריות, נשדרג קצת את מה שכתבנו בשלב הקודם, כדי לאפשר אחידות ותכנון הנדסת תוכנה נכון. מעתה, הפעלת פונקציות כמו הכפלה בסקלר וחיבור וקטורים יחזירו וקטור חדש במקום לבצע את השינוי על המופע שדרכו הופעלה הפונקציה כלומר יחזירו וקטור ולא void.

נציג את חתימות הפונקציות אותן שינינו:

במחלקה Coordinate: public Coordinate add(Coordinate other)

public Coordinate subtract(Coordinate other)

במחלקת Vector:

public Vector add(Vector vector)

public Vector subtract(Vector vector)

public Vector scale(double scalingFactor)

public Vector normalize() throws ArithmeticException

במחלקת Point3D:

public Point3D add(Vector vector)

public Point3D subtract(Vector vector)

כעת נכתוב את המחלקות החדשות.

ראשית, ניצור חבילה שנקרא לה Geometries.

חבילת Geometries

מחלקת Geometry: מחלקה אבסטרקטית שממנה יורשות כל הצורות הגיאומטריות.

יש בה getters ו-setteres של צבע הצורה והבהירות שלה. בנוסף יש גם פונקציה אבסטרקטית getNormal שמחזיקה את הנורמל לצורה.

מחלקת RadialGeometry: מטרתה לייצג צורות עגולות במרחב, ולכן היא יורשת מ-Geometry ויש לה בנוסף גם שדה של רדיוס. בעלת constructors, get ו-set לרדיוס.

אחרי שהגדרנו את הבסיס של צורות במרחב, נתחיל ליצור את המחלקות שמתארות ממש את הצורות. בפרויקט זה נשתמש בצורות בסיסיות ואיתן נוכל ליצור צורות מורכבות יותר. הצורות שבהן נשתמש יהיו כדורים, משולשים ומשטחים אינסופיים.

מחלקת Triangle:

יורשת גם מ-Geometry ומתארת משולש במרחב.

בעלת 3 שדות שמתארות את הקואורדינטות של הנקודות שמתארות את גבולות המשולש במרחב, בעלת contructors, getters ו-setters לכל אחת מהנקודות.

ופונקציית getNormal שמטרתה להחזיר את וקטור הנורמל למשולש.

בעלת פונקציה getIntersections: שמטרתה למצוא את נקודות החיתוך של הקרן הנורית מה-view plane עם המשולש. היא מקבלת כפרמטר קרן ומחזירה רשימה של Point3D שמהוות נקודות החיתוך של הקרן עם המשולש.

היחס בין הקרן למשולש יכול להיות אחד מהבאים:

1) הקרן לא נחתכת עם המשולש כיוון שהיא מקבילה לו ונחזיר רשימה ריקה.
2) הקרן לא נחתכת עם המשולש כיוון שהיא לא עוברת דרכו ונחזיר רשימה ריקה.
3) הקרן חותכת את המשולש פעם אחת, ונחזיר רשימה עם נקודה אחת.
4) נבצע מכפלה וקטורית על 2 מצלעות המשולש בשביל לקבל את הנורמל למישור בו המשולש נמצא וניקח אחד מקודקודי המשולש על מנת לייצג את המישור. לאחר מכן נבצע חיתוך של הקרן עם המישור. אחרי שמצאנו נקודת חיתוך, אם בכלל, נבדוק האם הן בכלל נמצאות בתוך תחומי המשולש או מבחוץ לו.

לאחר מכן עבור כל זוג קודקודים במשולש נחשב את הערכים הבאים:

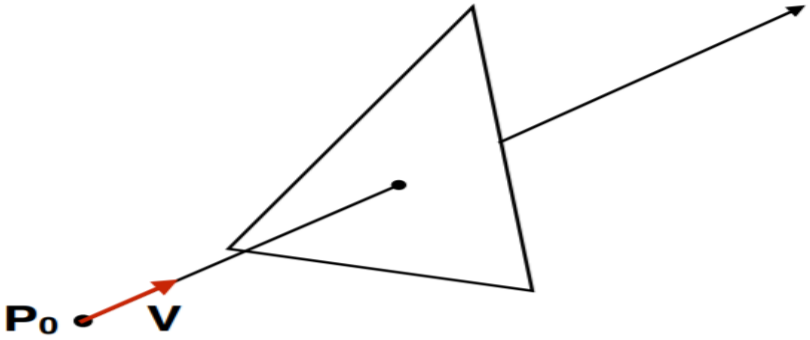
V1=T1-P0

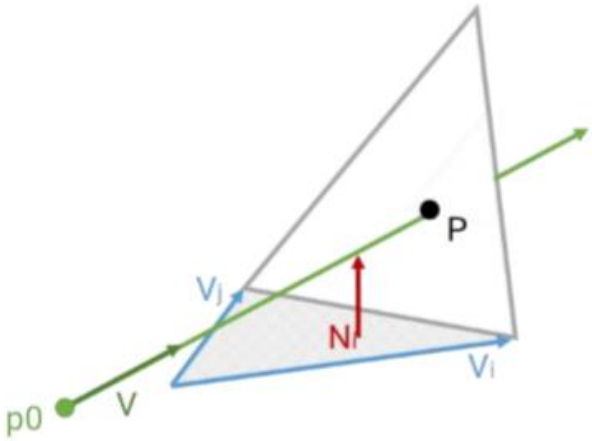
V2=T2-P0

Ni=V1xV2/|V1xV2|

לאחר מכן נבדוק עבור כל Ni*(P-P0) את הסימן שלו בעזרת פונקציית העזר sign . אם כל הסימנים זהים אזי הנקודה נמצאת בתוך המשולש וחזיר רשימה עם אותה נקודה. אם לא אזי הנקודה לא בתוך המשולש, ואז נחזיר רשימה ריקה.

ופונקציית עזר sign שמקבלת מספר וקובעת עם הוא חיובי או שלילי.





$$V_1=T_1-P_0$$
$$V_2=T_2-P_0$$
$$N_i = \frac{V_1 \times V_2}{|V_1 \times V_2|}$$

המחלקה Plane:

יורשת מ-Geometry, מייצגת מישור במרחב.

בעלת 2 שדות: נורמל משטח ונקודה על המשטח.

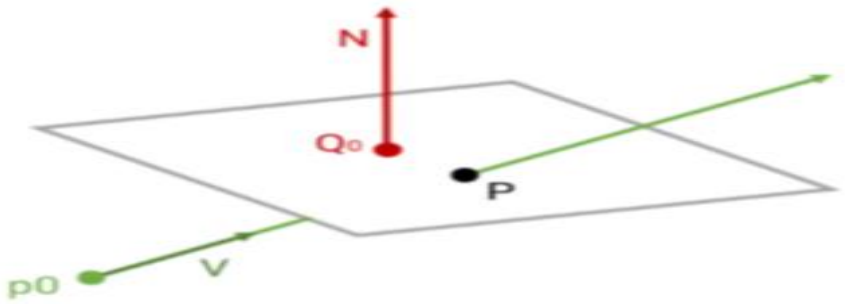
בעלת constructors ו-getters ו-setters על הנורמל והנקודה.

בעלת פונקציה findIntersections שמוצאת את החיתוך של קרן עם המישור. היא מקבל כפרמטר קרן ומחזירה רשימה של Point3D שמהוות נקודות חיתוך של הקרן עם המישור.

היחס בין הקרן למישור יכול להיות אחד מהמצבים הבאים:

1. הקרן לא חותכת כלל את המישור כיוון שהיא מקבילה אליו , ובמקרה זה נחזיר רשימה ריקה.
2. הקרן חותכת את המישור בדיוק פעם אחת, במקרה זה הפונקציה חזיר רשימה עם נקודה אחת.
3. הקרן לא חותכת את המישור כיוון שהיא מצביעה לכיוון השני ולא לכיוון המישור, אך לא נתעסק בקורס במקרה זה.

חישוב נקודת החיתוך יתבצע באופן הבא: ידוע שהווקטור בין הנקודה P לנקודה Q0 מאונך לנורמל למישור ולכן מכפלה סקלרית ביניהם תיתן 0. אל הנקודה P נוכל להגיע באמצעות הכפלת הוקטור V בסקלר t וחישוב הקו האנכי לנורמל. נכפיל סקלרית ונשווה ל-0: $N\cdot(P0 + t\cdot V - Q0) = 0$ ובאמצעות העברת אגפים נמצא את הערך t. אם t גדול מ-0 אזי קיימת נקודה ונוסיף לרשימה את הנקודה P0+tV ונחזיר את הרשימה.



המחלקה sphere: מתארת מעגל תלת מימדי במרחב. יורשת מ-radialgeometry ולכן יש לה רדיוס. בנוסף יש לה נקודה שמייצגת את מרכז המעגל.

יש לה constructors, getters ו-setters לנקודה ולווקטור הנורמל.

הפונקציה findIntersections כמו במחלקות הקודמות, מקבלות קרן כפרמטר ומחזירה רשימה של נקודות בהן הקרן חותכת את הצורה.

היחס בין הקרן למעגל יכול להיות אחד מהבאים:

1) הקרן לא חותכת כלל את הכדור ונחזיר רשימה ריקה.

2) הקרן משיקה לכדור, ולכן יש להחזיר את נקודת ההשקה, ונחזיר רשימה עם נקודה אחת בלבד.
3) נקודת ההתחלה של הקרן נמצאת בתוך המעגל ולכן יש נקודת חיתוך אחת עם המעגל ולכן נחזיר רשימה עם נקודה אחת.
4) נקודת ההתחלה של הקרן היא מחוץ למעגל והיא תחתוך את המעגל פעמיים.

האלגוריתם כדלהלן: ניצור וקטור בשם L שיוצא ממרכז הכדור O אל נקודת הראש של הקרן P0 ע"י החיסור הבא: L=O-P0. לאחר מכן נחשב את אורך הקטע tm שהוא היטל הווקטור L על ווקטור הכיוון של הקרן v ע"י מכפלה סקלרית ביניהם: tm=V*L

נחשב את d שהיא הניצב במשולש הנוצר ע"י הווקטורים L ו-V באמצעות פיתגורס:

$$tm = \sqrt{V^2 - tm^2}$$

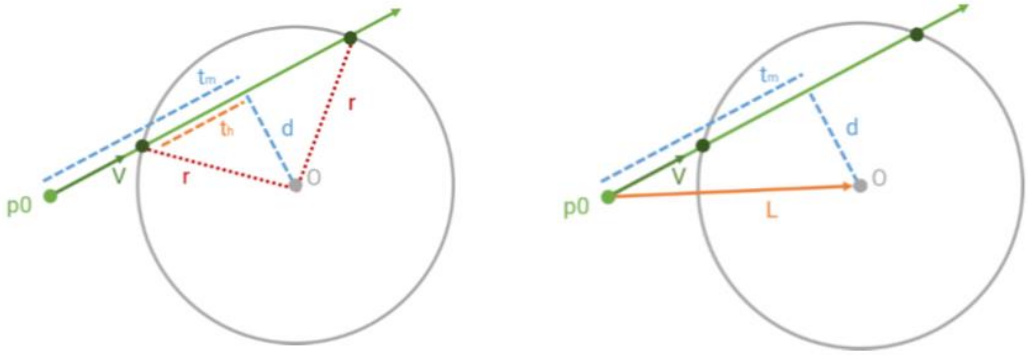
כעת נוכל לקבוע האם בכלל יש נקודות חיתוך עם הכדור. במידה ו-d גדול יותר מהרדיוס של המעגל, משמעות הדבר היא שהקרן בכלל עוברת מחוץ לכדור, ולכן במצב כזה נפסיק את הפונקציה ונחזיר רשימה ריקה. נחשב את אורך הקטע th ע"י משפט פיתגורס במשולש הנוצר ע"י הקטע d והרדיוס r:

$$th = \sqrt{r^2 - d^2}$$

האורך th הוא למעשה המרחק בין נקודת החיתוך של d עם הקרן. לכן על מנת למצוא את נקודות החיתוך עם הכדור נחשב שני ערכים חדשים:t1=tm-th הוא הערך של הנקודה הקרובה ביותר ל-P0 והערך t2=tm+th הוא הערך שרחוק יותר מ-P0.

עבור כל אחד מערכי ה-t החדשים שחישבנו נחשב את הנקודה המתאימה לו על פני הכדור (רק אם ti גדול מ0) באופן הבא: Pi=P0+ti*V.

נוסיף את הנקודות שמצאנו אל רשימת הנקודות ונחזיר אותה.



המחלקה Light:

מחלקה שממנה כל סוגי התאורות יורשות ממנה, בעלת שדה color(צבע), בנאים, ופונקציה getIntensity שמחזירה את הצבע עם העוצמה המתאימה.

המחלקה ambient Light: מייצגת את התאורה הסביבתית, שהיא למעשה תאורה כללית של הסצנה. בעלת שדה ka שמבטא את העוצמה של התאורה. יש לה בנאים ו-getters ו-setters לעוצמה ולצבע, כמו כן getIntensity שמחזיר את הצבע מוכפל בעוצמה.

בשביל לייצג סצנה גרפית הכוללת מגוון צורות גיאומטרית, מצלמה, תאורה סביבתית ועוד תאורות בעתיד נצטרך לבנות מחלקה חדשה שתכיל את כל העצמים יחד שתהיה הסצנה שאותה יהיה נין לרנדר בהמשך.ניצור מחלקה כזאת בשם Scene בתוך חבילה חדשה בשם scene.

המחלקה Scene:

מהווה את הסצנה הגרפית בה נמצאים כל העצמים.

השדות הם: שם הסצנה, צבע הרקע, תאורה סביבתית, מצלמה שהכל מוצג ביחס אליה והיא מגדירה את זווית הצילום של הסצנה,screenDist-שמייצג את מרחק הסצנה מהמצלמה,list מסוג geometry שמכילה את כל הצורות הגיאומטריות הנמצאות בסצנה. עובד על עיקרון הפולימרפיזם בכך שמחלקת האב של כל הצורות הוא geometry, list מסוג lightSource שמייצג את כל מקורות האור שהן לא תאורה סביבתית. בשלב 2 לא נשתמש ברשימה זו.

המחלקה מכילה בנוסף get ו-set לשדות.

פונקציית addGeometry שמוסיפה צורה חדשה לרשימת הצורות הנמצאות בסצנה.

פונקציית getGeometriesIterator המחזירה איטרטור לרשימת הגופים בסצנה, על מנת לאפשר מעבר איטרטיבי על הגיאומטריות של הסצנה. יש לנו גם פונקציות זהות לשניים האחרונות גם למקורות האור אך נרחיב עליהן בשלב 3.

מחלקת Camera:

מטרתה להוות ציר היחס של הסצנה, הסצנה נראית ביחס אליה.

השדות הן: P0-נקודת המרכז של המצלמה. vUp- הווקטור כלפי מעלה,vTo- הווקטור לכיוון הסצנה, vRight-ווקטור לכיוון ימינה, מתקבל כתוצאה ממכפלה וקטורית של vTo ו-vUp.

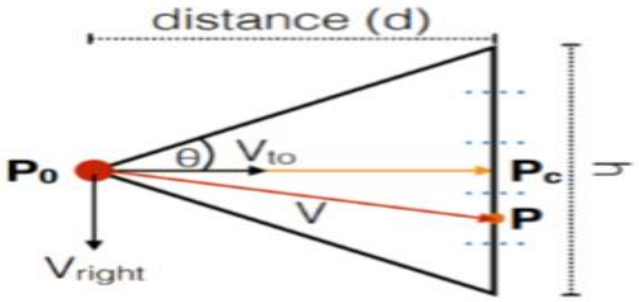
מכילה בנאים ופונקציות get ו-set על הווקטורים כאשר שינוי של אחד מהווקטורים יביא לשינוי אוטומטי של vRight.

פונקציית toString שמכילה פירוט של כל השדות המחלקה.

פונקציית constructRayThroughPixel שמקבלת את מספר הפיקסלים בציר x ובציר y ואת האינדקס בציר x ובציר y, המרחק של המצלמה מה-view plane , גובה המסך ורוחב המצלמה והיא מחזירה את הקרן שיוצאת מהמצלמה ופוגעת באחד מהפיקסלים ב- view plane.

החישוב נעשה כך:

$$P = P_C + \left[\left[\left(x - \frac{\#\,pixels_x}{2}\right)R_x + \frac{R_x}{2}\right]V_{right} - \left[\left(y - \frac{\#\,pixels_y}{2}\right)R_y + \frac{Ry}{2}\right]V_{up}\right]$$



ולאחר מכן נחזיר את הקרן שהיא P-P0.

נוסיף מחלקה נוספת האחראית על כתיבת הפיקסלים לתמונה.

נכתוב אותה בתוך חבילה חדשה בשם renderer.

המחלקה imageWriter:

מטרתה להגדיר את תכונות המסך(התמונה) שתוצג.

בעלת שדות הבאות: imageWidth- רוחב התמונה, imageHeight- גובה התמונה, Ny- מספר הפיקסלים בציר ה-y, Nx- מספר הפיקסלים בציר ה-x. Project_path- מיקומה הגרפי של התמונה. image_- באפר המסוגל לאחסן פיקסלים במבנה של מערך דו ממדי ולאחר מכן לתרגם אותם לתמונה על הזיכרון של המחשב. imageName_- שם התמונה.

מכילה בנאים, ופונקציות Get ו-Set לשדות.

פונקציית writeToImage שמטרתה לכתוב את התמונה למקום הפיזי בזיכרון המחשב.

פונקציות writePixel השונות שמטרתם לצבוע את הצבע המתאים בפיקסל המתאים.

מחלקת renderer:

מטרתה היא להיות המנוע הגרפי של התוכנית ולקחת את כל האלמנטים ולעשות מהם סצנה דה-פקטו.

בעלת השדות הבאות:scene_- אובייקט סוג סצנה שמכילה את כל הצורות והאלנטים, imageWriter_-שכותב את הצבע בפיקסל המתאים.

מכילה בנאים. מכילה פונקציית renderImage שעוברת על כל פיקסל במסך וצובעת אותו לפי הצבע של הצורה הראשונה בה הקרן פגעה בתנועתה בתוך הסצנה.

פונקציית getSceneRayIntersections: מקבל קרן ומחזיר את כל הצורות ואת כל הנקודות שהקרן חותכת דרכם.

פונקציית getClosestPoint: מקבל map שלמעשה מכיל רשומות על צורות ואת נקודות החיתוך איתן ומחזיר Map שמכיל רשומות שבכל רשומה יש את הצורה ואת הנקודה שבה הקרן חתכה את הצורה לראשונה.(הנקודה הכי קרובה למקור הקרן).

פונקציית calcColor: מקבל נקודה ומחזיר את הצבע בעוצמתה המתאימה.

פונקציית writeToImage: מפעיל את הפונקצייה במחלקה imageWriter שמטרתה לכתוב את התמונה לזיכרון פיזי על המחשב.

פונקציית printGrid: מקבלת int כפרמטר שמייצג את גודל הריבוע של הרשת הנפרשת על פני התמונה בפיקסלים.

פונקציית calcColor: מקבלת map עם רשומה של geometry ו Point3D ומחזירה את הצבע בנקודה.

שלב 3- תאורות

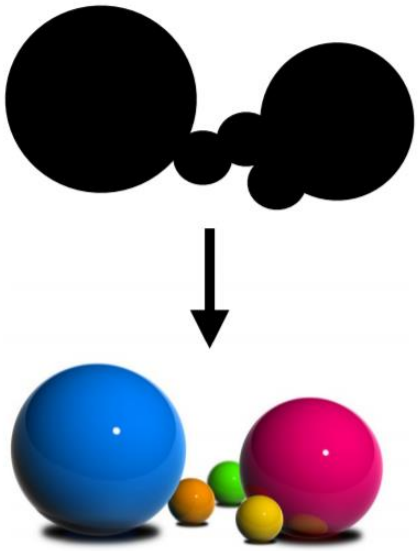
בשלב זה נוסיף את התאורות לפרוייקט בכדי ליצור תמונה בעלת תאורות שונות ובכך ליצור תמונה ריאליסטית יותר.

ניעזר ב-5 מחלקות: light,ambient light,directiona light,point light, spot light.

Light: מחלקת האב שממנה כולם יורשים, אבסטרקטית,כלומר לא ניצור ממנה מופעים, בעלת שדה צבע(color) ופונקציית getIntensity שמטרתה להחזיר לנו את הצבע של האור בהכפלה יחד עם קבוע הנחתה כלשהו. מכילה גם בנאים (דיפולטיבי והעתקה).

Ambient Light: מספק ניראות בסיסית של הגיאומטריות בסצנה, משפיע על כל הצורות בצורה שווה.

יורש מ-light ולכן בעל שדה של צבע. בנוסף יש לו שדה של מקדם הנחתה(Ka).

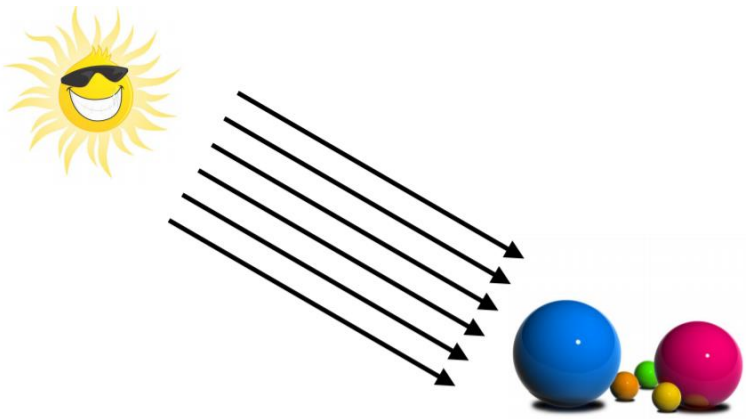


Directional Light: מייצג אור שמגיע מהאינסוף, כמו השמש. בעל תכונת כיוון.

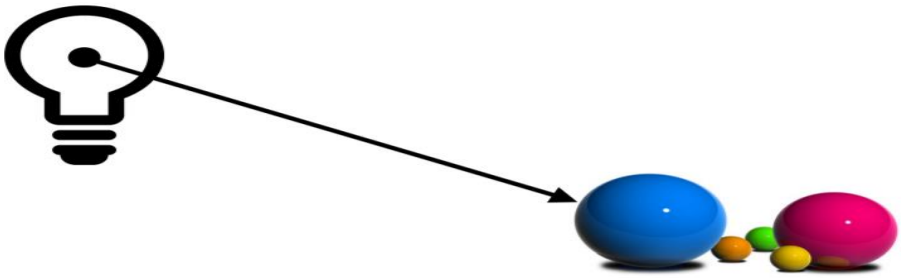
אין הנחתה, כלומר לא תלוי במרחק של מקור האור מהעצם.

יורש מ-Light, מכיל בנוסף שדה כיוון(direction).

מכיל בנאים, פונקציות get ו-set.



Point Light: מתאר מקור אור שהוא רב כיווני, כלומר בכל מקום, אין לו שדה כיוון.כמו מנורה.



יורש מ-light וממש את Light Source, יש לו בנוסף שדה מיקום (_position) שמתאר את המיקום של מקור האור במרחב ו3 מקדמים:kc,kl,kq . בעל בנאים, פונקציות Get ו-set.

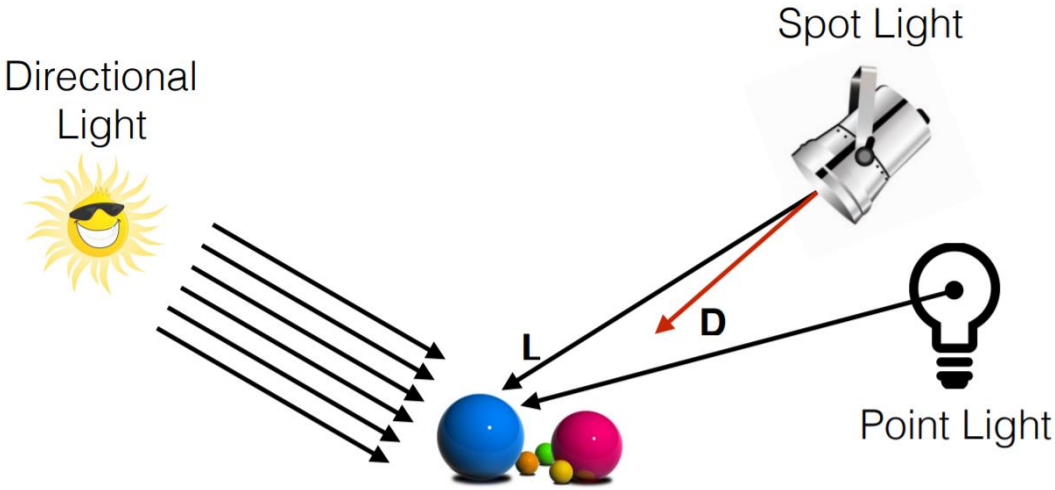
בעל פונקציית עזר setBoundary שבודקת שאין חריגה מגבולות הערכים של RGB, בחישוב האינטנסיביות של הצבע בפונקציה getIntensity.

החישוב נעשה באופן הבא:

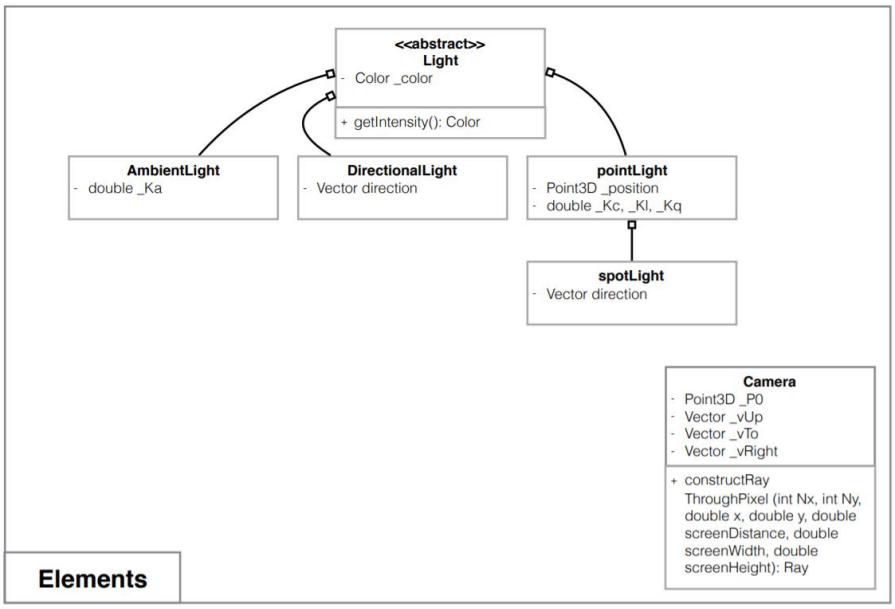
$$I_L = I_0 / (K_c \cdot k_j d \cdot k_q d^2)$$

Spot Light: מתאר למעשה point light עם כיוון. יורש ממחלקת point light ובעל שדה נוסף של כיוון (direction). מכיל בנאי ופונקציית getIntensity.

תיאור של האורות בסצנה:



ארכיטקטורת הירושות שלנו תיראה כך:

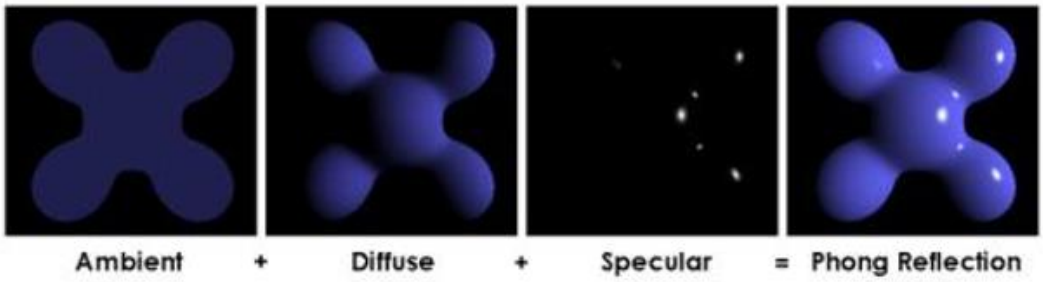


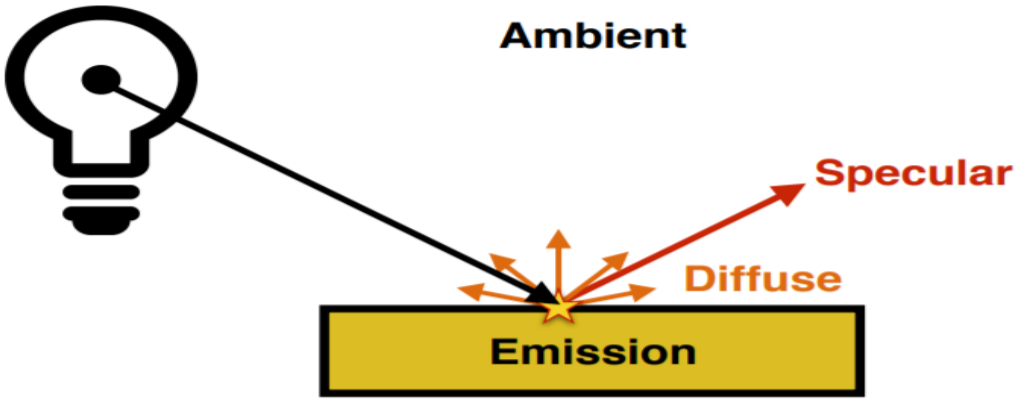
לאחר שהגדרנו את התאורות נסביר איך הכל מתחבר יחד.

לפי מודל ההארה של פונג לכל אובייקט יש החזרה מסוימת של אור כאשר האור פוגע בו.

החיבור בין תאורה סביבתית (ambient) להתפשטות האור (diffuse) והאור המוחזר (specular) משווה לנו מראה של אובייקט תלת ממדי.

נמחיש זאת ע"י התצלום הבא:





כעת נחשב את העוצמה של האור בנקודה מסוימת :

נוסיף את ההחזרה של האובייקט עצמו: Ipoint3D = IE

נוסיף את התאורה הסביבתית (לפי המשוואה): Ipoint3D = IE + KAMIAM

נוסיף את הפיזור של האור(DIFFUSE):Ipoint3D = IE + KAMIAM+KD(N*L)IL

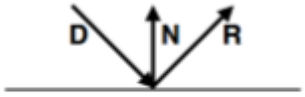
N ו- L ווקטורים מנורמלים כאשר L הוא קרן האור שפוגעת באובייקט ו-N הוא הנורמל לנקודה.

המכפלה הסקלארית בניהם נותנת את קוסינוס הזווית שביניהם. הדיפוזיה מרבית כאשר היא בכיוון הנורמל, KD הוא גורם ההרחקה וIL- הוא העוצמה שבנקודה

נוסיף את ההחזרה של האור (specular) Ipoint3D = IE + KAMIAM + KD(N*L)IL + KS(V*R)^nIL

Ks הוא גורם ההרחקה. V ו-R הם וקטורים מנורמלים, V הוא הווקטור מנקודת הצפייה באובייקט ו-R הוא הצפוה בסצנה שרואה את ההחזר של האור. המכפלה הסקלארית ביניהם נותנת את קוסינוס הזווית שביניהם, כלומר עד כמה הצופה בסצנה רואה את ההחזר של האור. החזרת אור מקסימלית היא בכיוון ההשתקפות. האור מצטמצם באופן אקפוננציאלי (חזקת n) ככל שהזווית בין נקודת הראייה של הצופה(v) ובין ווקטור ההחזרה של האור (R) גדלה. n הוא מידת המבריקות של האוייקט: ככל שהאובייקט בורק יותר, n יגדל בהתאמה ואז ההחזרה של האור תהיה גדולה יותר.

נחשב את R באופן הבא:



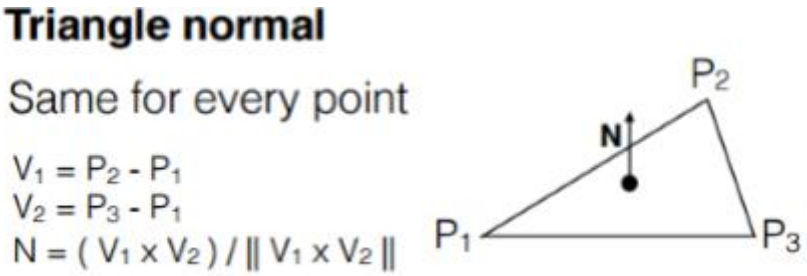
$R = D - 2(D \cdot N)N$

אם כן קיבלנו את הנוסחה המלאה לחישוב עוצמת הצבע בנקודה במרחב לפי מודל פונג:

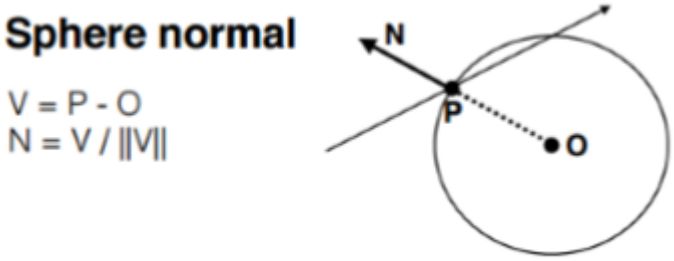
$$I_{point3D} = I_E + K_{AM}I_{AM} + K_D(N \cdot L)I_L + K_S(V \cdot R)^n I_L$$

מאחר ונצטרך לחשב עבור כל גיאומטריה את הנורמל לנקודה שעליה, נגדיר במחלקה Geometry שכל הגיאומטריות יורשות ממנה את המחלקה get Normal שמקבל נקודה ומחזיקה את הנורמל לנקודה זו.

עבור משולש:



עבור כדור:



.

שיפורים-

שיפור ראשון-superSampling:

המשמשת להסרת שיפור הגורם לטישטוש גבולות הצורה והפיכת התמונה לנקייה יותר משיברור. כלומר שיטה קצוות משוננים. בניגוד לאובייקטים בעולם האמיתי אשר יש עקומה חלקה ורציפה מסך המחשב מציג לצופה מסך גדול של ריבועים קטנים המכונים פיקסלים. לפיקסלים אלה יש את אותו גודל, ולכל אחד מהם יש צבע אחד. ולכן בקצוות בולט השינון שנוצר בין הרקע לצורה.

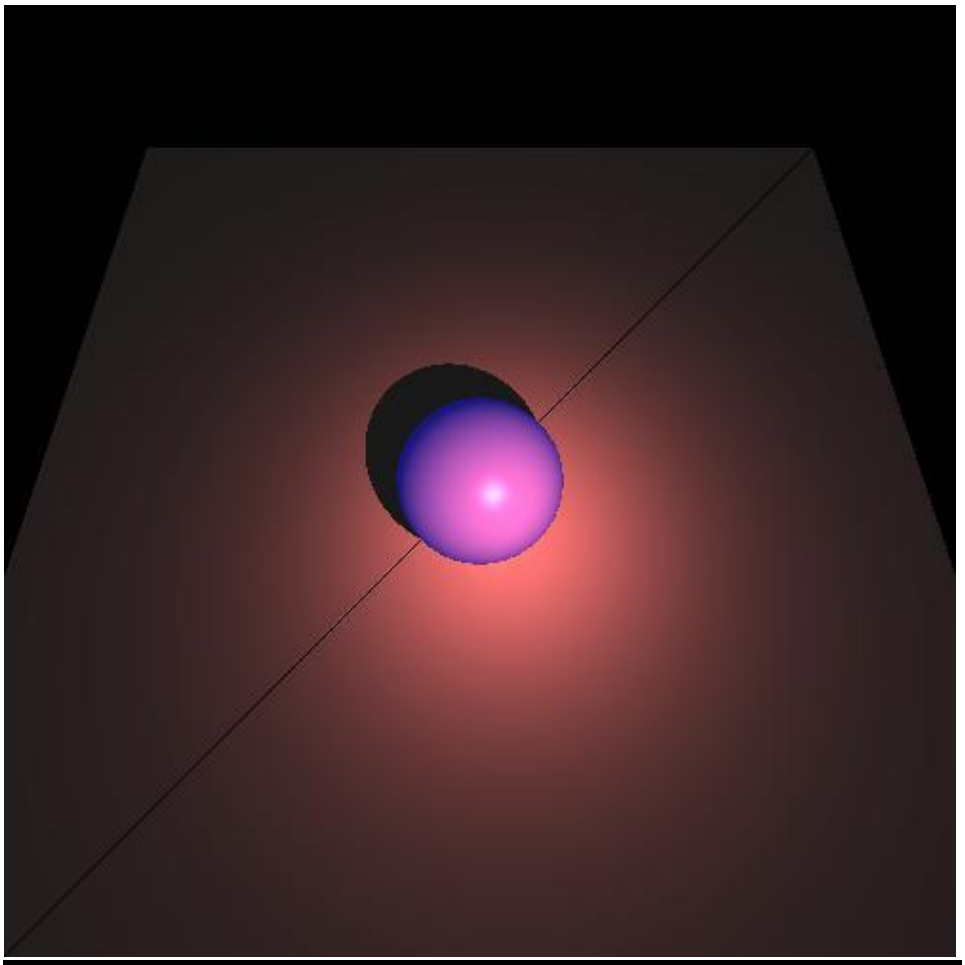
להלן הפונקציה שנוספה

```
    private Color superSampling(Ray ray,double space){
        Entry<Geometry,Point3D> entry;

        Color tempcolor=scene.getBackGround();
        ray.setStartPoint(ray.getStartPoint().addVector(new Vector(space,space,0)));
        Map<Geometry,List<Point3D>> intersections = getSceneRayIntersections(ray);
        if(!intersections.isEmpty()) {
                        entry = getClosestPoint(intersections).entrySet().iterator().next();
                        tempcolor=calcColor(entry.getValue(),entry.getKey(),ray,0);
                    }
        Color color=tempcolor;

        tempcolor=scene.getBackGround();
        ray.setStartPoint(ray.getStartPoint().addVector(new Vector(0,-4,0)));
        intersections = getSceneRayIntersections(ray);
        if(!intersections.isEmpty()) {
                        entry = getClosestPoint(intersections).entrySet().iterator().next();
                        tempcolor=calcColor(entry.getValue(),entry.getKey(),ray,0);
                    }
        color=mixColors(color,tempcolor);

        tempcolor=scene.getBackGround();
        ray.setStartPoint(ray.getStartPoint().addVector(new Vector(space*2,0,0)));
        intersections = getSceneRayIntersections(ray);
        if(!intersections.isEmpty()) {
                        entry = getClosestPoint(intersections).entrySet().iterator().next();
                        tempcolor=calcColor(entry.getValue(),entry.getKey(),ray,0);
                    }
        color=mixColors(color,tempcolor);

        tempcolor=scene.getBackGround();
        ray.setStartPoint(ray.getStartPoint().addVector(new Vector(0,space*2,0)));
        intersections = getSceneRayIntersections(ray);
                    if(!intersections.isEmpty()) {
                        entry = getClosestPoint(intersections).entrySet().iterator().next();
                        tempcolor=calcColor(entry.getValue(),entry.getKey(),ray,0);
                    }

        return mixColors(tempcolor,color);
```

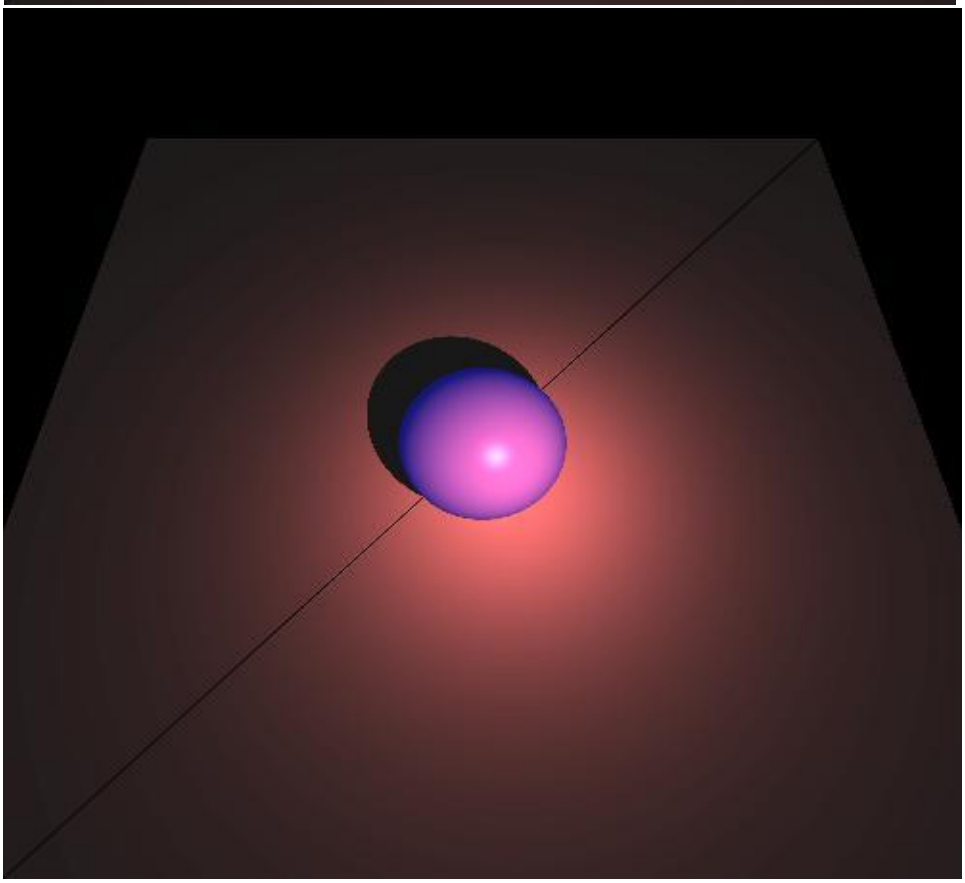
מראה המבחנים לפני ואחרי השיפור:

לפני:

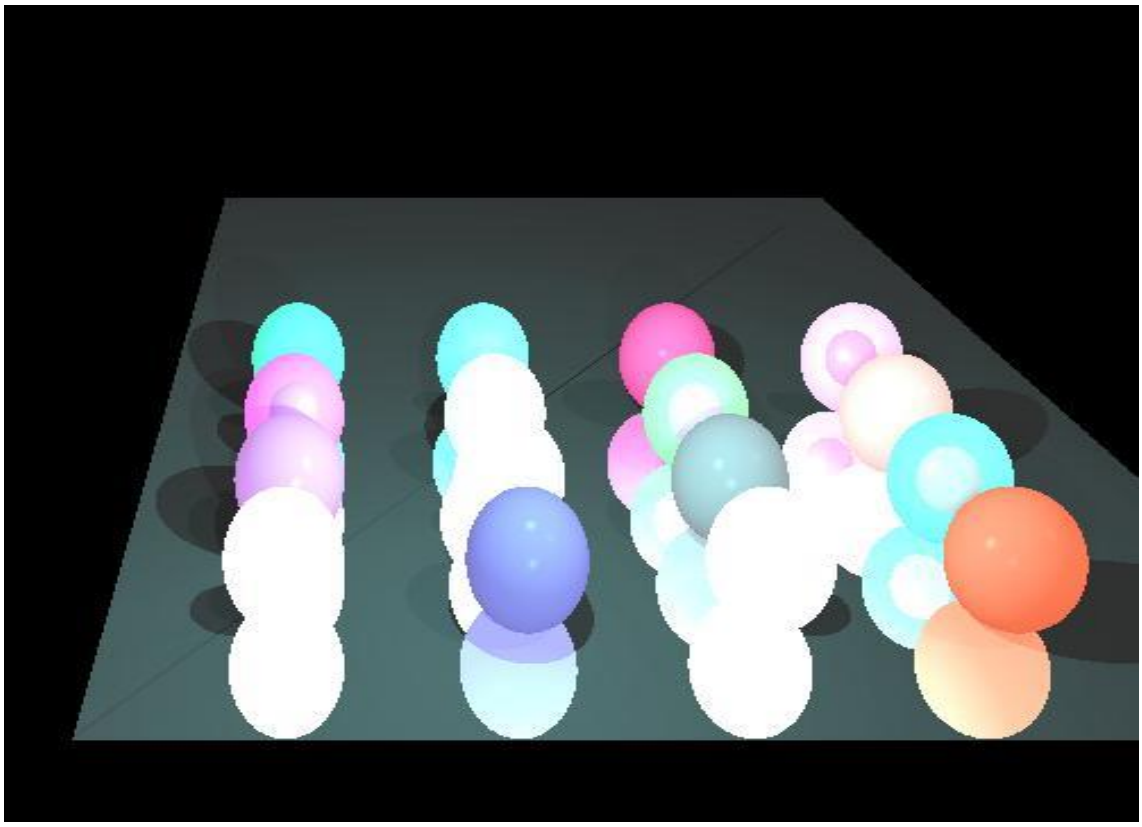
:



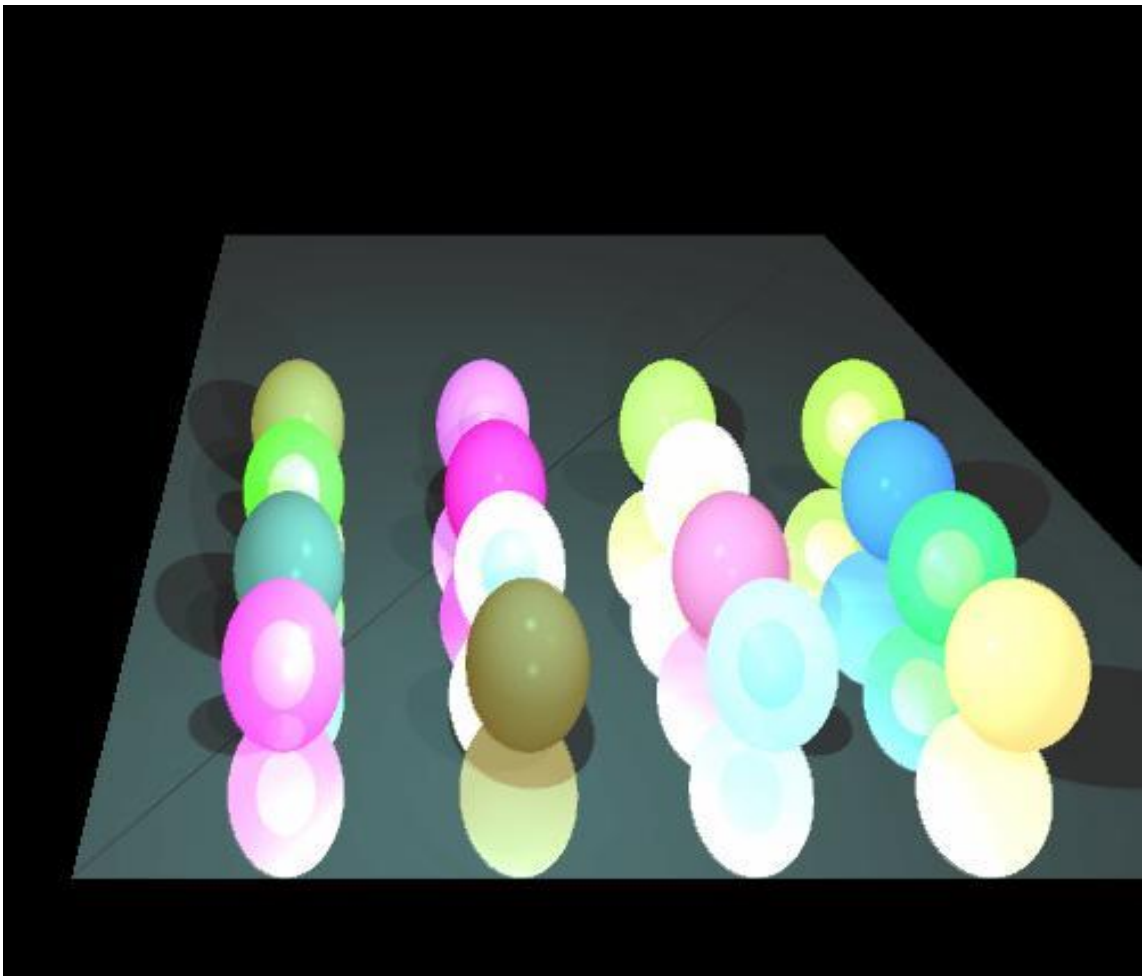
אחרי:



לפני:



אחרי:



שיפור שני- שיפורי זמן ריצה:

השיפור שעשינו כדי לטשטש את גבולות הצורות בתמונה, גרר שליחת קרניים נוספות על גבי כל פיקסל מה שהוסיף לזמן הריצה לא מעט. השיפור שעשינו הינו חלק מהפונקציה הרינדור של התמונה ומטרתו לחסוך שליחת קרניים כאשר התמונה מזהה שהגענו לפיקסל ללא נקודות חיתוך.

```
91
92          if(intersections.isEmpty()) {
93              /**
94               * if the middle of the pixel is empty than dont calculate the supersampling
95               * and insert the background color- improvement in rendering time
96               */
97              imageWriter.writePixel(j, i, scene.getBackGround());
98          }
99          else {//If there are any cut points then you will return the nearest crop point.
.00             Entry<Geometry,Point3D> entry=getClosestPoint(intersections).entrySet().iterator().next();
.01             Color color=calcColor(entry.getValue(),entry.getKey(),ray, level: 0);
.02             /**
.03              * calculating the average color of he pixel by getting the supersamling color from its functi
.04              * with how much you want to go to the sides for each sample.
.05              * the call for supersamling is inside the else to save computing time
.06              * so when the center of the pixel is a background dont calculate the supersampling
.07              */
.08             color=mixColors(color,superSampling(ray, space: 1));
.09
.10
.11             imageWriter.writePixel(j, i, color);//Request from imageWriter to write a certain color to the
.12         }
```

הערות לאורך הפרויקט:

לאורך כל הפרויקט כל פונקציה קיבלה הסברים מפורטים המכילים את שם הפונקציה הפרמטרים שלה מה היא מחזירה ומה היא עושה. מבנה ההערה נראה כך:

```
/**************************************************
 * FUNCTION:
 renderImage.
 * PARAMETERS:
 null.
 * RETURN VALUE:
 null.
 * MEANING:
 A function that builds a ray for each pixel and checks for Intersection points with shapes in the scene.
 **************************************************/
```

בתוך חלק מהפונקציה איפה שהרגשנו שיש צורך נכתבו גם הערות פנימיות שעוזרות להבנת הפונקציה.